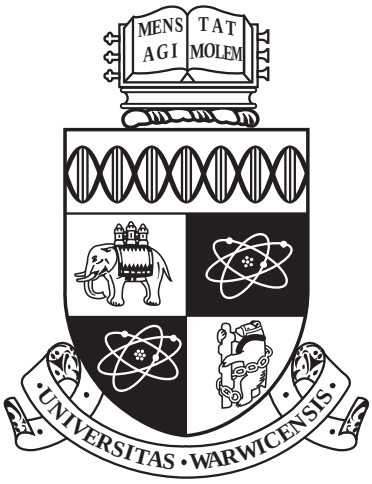

Advances in Computational Methods for Transmission Electron Microscopy Simulation and Image Processing

by

Mark Adam Dyson

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

Department of Physics

March 2014

THE UNIVERSITY OF
WARWICK

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank all the members of the Microscopy Group at Warwick who have helped and encouraged me throughout the duration of my PhD and made my time at Warwick a memorable experience. I would especially like to thank Reza Kashtiban for patiently helping me during the trial-and-error phase of my microscope software development, and Ana Sanchez for the long hours spent on the microscope providing me with the copious amounts of data without which I would not have been able to develop my EWR software. I would also like to thank my supervisors Jeremy Sloan and Neil Wilson for all of their support and guidance and for the oppportunites they have given me along the way.

Declarations

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree.

The work presented (including data generated and data analysis) was carried out by the author except in the cases outlined below:

- Image series of fluorinated graphene in chapter 5 were acquired by Reza J. Kashtiban of The University of Warwick using the acquisition script developed by myself (detailed in section 4.2). Diffraction patterns of fluorinated and pristine graphene were also acquired by Reza J. Kastiban.
- Image series of polymersome and cylindrical miceles in chapter 6 were acquired by Ana M. Sanchez of The University of Warwick using the acquisition script developed by myself (detailed in section 4.2).
- Density Functional Theory potentials used for the simulation of fluorinated graphene (fig. 5.15) and defects in graphene (fig. 3.2) were calculated by Peter Brommer of The University of Warwick.

Some of this work has been previously published in the following joint publications:

1. Dyson, M. A., Sanchez, A. M., Patterson, J. P., O'Reilly, R. K., Sloan, J., and Wilson, N. R. (2013). *A new approach to high resolution, high contrast electron microscopy of macromolecular block copolymer assemblies.* Soft Matter, 9(14), 3741-3749. doi:10.1039/C3SM27787A
2. Kashtiban R., Dyson M. A., Nair R., Zan R., Wong S. L., Ramasse Q., Geim

A., Bangert U., and Sloan J. (forthcoming). *Atomically Resolved Imaging of Highly Ordered Alternating Fluorinated Graphene.* Nature Communications.

The software developed in this work and used to perform all multislice simulations (unless otherwise indicated) has now been made available open-source at https://github.com/ADyson/CUDAMultislice.

Abstract

Modern electron microscopes are fitted with ever larger charge-coupled device (CCD) cameras capable of faster acquisition rates which in turn drives a concomitant increase in the bandwidth of data that is being collected and the amount of information in our datasets. At the same time, current increases in computational performance are largely being delivered through the addition of parallel execution units rather than explicit increases in the speed of single processors, this means techniques that cannot exploit their inherent parallelism are seeing little performance benefit from the generational improvements in computer processors. Many techniques used in electron microscopy to process these large datasets have not been adapted to utilise the modern methods available for parallel data processing which can lead to lengthy offline data processing techniques which could otherwise be performed in near real-time. Reimagining these methods to suit highly parallel computational architectures such as graphics processing units (GPUs) can offer improved performance orders of magnitude higher than their central processing unit (CPU) counterparts.

In this thesis I have looked specifically at the case of transmission electron microscopy (TEM) image simulation via the multislice procedure, and exit wave reconstruction (EWR), which can both potentially see huge benefits by adapting these algorithms to exploit their parallelism. Software has been developed for performing multislice simulations using GPU computation where the increase in computational power also allows for modifications to be made which can increase the accuracy of the simulations at the expense of simulation time. The multislice software developed here has no minimum slice thickness limitations and the slice thickness no longer has to be coupled to the structure being simulated to ensure accuracy. The CCD detector characteristics and electron dose have also been incorporated within the simulation process. The use of GPUs has allowed these simulations to be performed in vastly less time than CPUs based equivalent simulations.

Software has also been developed for performing EWR on either multicore CPUs or GPUs which lowers the time required to perform EWR sufficiently that real-time reconstruction at typical CCD frame-rates is a distinct possibility. This EWR software additionally features mutual information (MI) based image alignment which can handle accurate image alignment in cases where other methods are prone to failure.

These software are used to aid in the investigation of fluorinated graphene conformation via multislice simulation and EWR, and in the study of self-assembled block co-polymer assemblies also by EWR.

Chapter 1

Introduction

1.1 Introduction

Since it's invention in 1932 [Knoll and Ruska 1932], TEM has been one of the leading techniques for understanding objects on the nanoscale. Traditional light microscopy is resolution limited at half the wavelength of light, typically around $0.25\mu m$. The wavelength of electrons is dependent upon the voltage to which they are accelerated and can be made much smaller than the wavelength of light by acceleration to a high voltage. The wavelength of electrons at a typical TEM operating voltage of $200kV$ is around $0.0025nm$, theoretically small enough to easily resolve the individual atoms within a structure, there are also electron microscopes capable of accelerating electrons well past $1MV$ but the electron wavelength is not the only factor in determining the achievable resolution [Thomas and Lacaze 1973].

Unfortunately the electromagnetic lenses required to focus an electron microscope are still at a much poorer standard than those used to focus light in an optical microscope, additionally it can be difficult to produce a bright monochromatic electron source so electron microscopes are far from reaching the resolution limit that would be imposed by the electron wavelength at these voltages. Adjusting the electron wavelength can also have potential negative consequences for beam induced specimen damage which can favour the use of lower voltages, this has lead to to a focus on methods that can improve the quality of imaging through a variety of other means. The current generation of electron microscopes are typically fitted with aberration correctors and field emission electron sources which enables them to routinely achieve atomic resolution for many different kinds of samples but there is still significant room for improvement.

This particular study focuses on improving the image reconstruction methods which are used to improve the quality of electron microscopy images through a combination of software and hardware, and also on improving the computational methods used to simulate TEM images from computer generated structures for a

variety of microscope conditions.

1.2 Introduction to Transmission Electron Microscopy

A basic transmission electron microscope consists of an electron producing source which may be a tungsten hairpin, a lanthanum hexaboride (LaB_6) source or a field-emission gun, and a series of lenses which can be used to focus the electrons onto a sample, and then to focus the electrons into an image or diffraction pattern on a viewing screen or camera after interaction with the sample, a schematic overview of this is given in fig. 1.1 although an actual TEM would contain far more lenses than just those depicted here. In a modern TEM there may also be a monochromator to increase the coherency of the electron source, as well as separate aberration correctors for both the electron probe and imaging system in addition to large numbers of additional detectors dedicated to collecting secondary signals such as electron-energy loss and x-ray emission etc. [Williams and Carter 1996].

1.2.1 The Theory of Image Formation

The description of a TEM can be roughly divided in two main sections, the illumination system which controls how the electrons arrive at the sample, and the imaging systems which controls how the transmitted electrons are displayed at the viewing screen/CCD camera. The TEM can also be operated in numerous ways depending on the kind of data that is required. For the TEM mode of operation the role of the illumination system is to take the electrons from the source and produce an approximately parallel beam at the specimen with enough intensity to produce a useful signal later on. This parallel beam is then refocused into an image or diffraction pattern at the viewing screen. For scanning transmission electron microscopy (STEM) operation, a focused probe is formed instead of a parallel beam to generate signal from a very specific area, to generate an image the probe has to be scanned across an area of the sample to observe the change in the produced signal as a function of probe position.

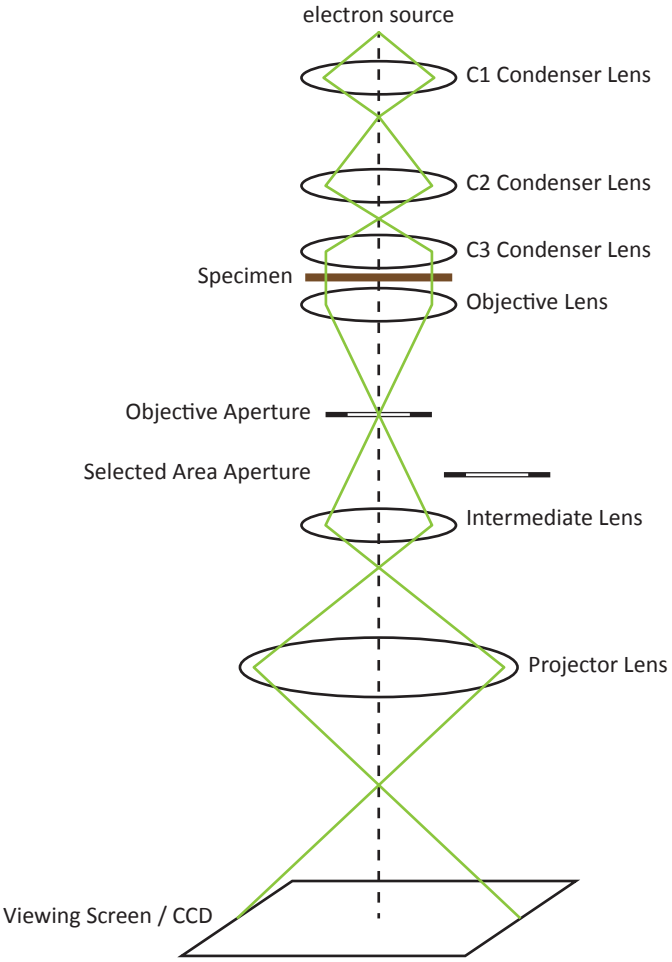

Figure 1.1: Schematic overview of a basic TEM showing relative positions of lenses and apertures and the specimen and viewing screen.

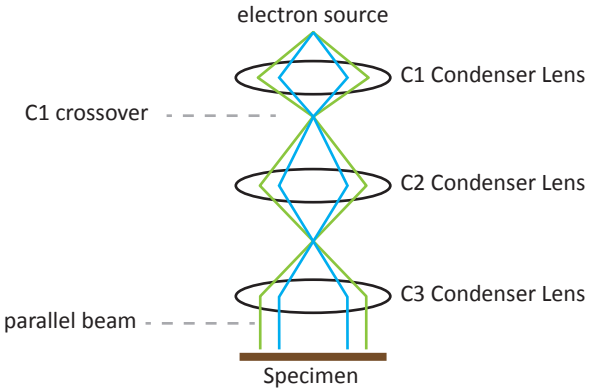

Figure 1.2: Schematic overview of the illumination system in a TEM for standard imaging under approximately parallel beam conditions.

In TEM operation mode, after the parallel beam of electrons passes through and interacts with the specimen (fig. 1.2), the objective lens is used to disperse the electrons to form a diffraction pattern in its back focal plane or an image in its image plane as depicted in fig. 1.3 where k is the spatial frequency of the scattered electrons. The projector lens system then selects either the back focal plane or image plane of the objective lens as its object in order to display the image or diffraction pattern on the viewing screen or CCD. This setup also ensures there are other image planes conjugate to the specimen plane in which apertures can be inserted for selected area electron diffraction (SAED) to control the specimen area that contributes to a diffraction pattern. An objective aperture can also be inserted into the back focal plane of the objective lens where the diffraction pattern is formed, this can be used to select which scattered electrons (i.e. just the central beam, just a scattered beam, or a large number of beams) are used to form the image.

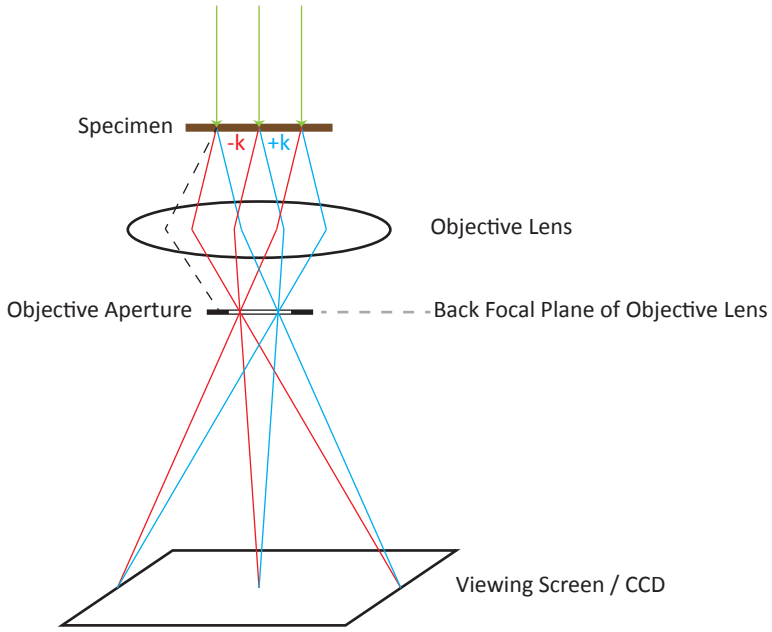

Figure 1.3: Schematic overview of the objective lens in a TEM. At the back focal plane beams diffracted from the specimen with equal angle are all focused to the same spot, and at the image plane, diffracted beams from the same location are focussed to the same point. The objective aperture prevents beams with large scattering angle from reaching the image plane. The projection system used to magnify the image has not been included.

1.2.1.1 Phase Contrast Imaging

When the objective aperture in the back focal plane of the objective lens is used to select more than just a single beam to contribute to the image, then these

conditions can be used to achieve phase contrast imaging. In this mode of operation image contrast is formed primarily as a result of differences in phase between the electrons that have been scattered by different amounts. This mode of imaging can be incredibly sensitive to a large number of factors, including, but not limited to specimen thickness, specimen orientation, microscope focus, microscope astigmatism and objective aperture size. For high resolution TEM work many other higher order aberrations have an increasingly important role in the contrast formation (described in section 1.2.1.2 etc.). The complicated nature of contrast formation in phase contrast imaging is largely the reason why image simulation is such an important tool for high resolution transmission electron microscopy (HRTEM), the contrast in an image is often difficult to predict based purely on atomistic knowledge of the structure (no direct relationship) and varies so strongly with the imaging conditions that simulations are required to understand what is being seen experimentally.

To mathematically describe the process of phase contrast imaging we start by describing the incoming parallel beam of electrons by a plane electron wave parallel to the optic axis of the microscope. As this plane wave passes through the sample the electrons will be scattered by the structure (described in more detail later) and at the exit-plane of the specimen we will have an electron wave which has been modified, and these modifications carry the information about the structure, this is called the *exit-plane wavefunction*. If we assume that the object that we are imaging is a phase object, this means its primary effect on the electron wave is to change only its phase, and not its amplitude. The phase modulation of the electron wave passing through the specimen can be written as $\exp -i\phi(r)$, here $\phi(r)$ is the object function which describes the effects of the specimen on the electron wave, $\phi(r)$ is related to the projected electrostatic potential of the sample $V_p(r)$ [Williams and Carter 1996].

This phase modulation by the projected potential of the sample can be visualized by imagining two electrons passing through the sample at different locations, a first electron which passes though a potential $V_{p1}(r_1)$ will gain potential energy eV_{p1} thereby temporarily reducing its wavelength, a second electron which passes through a different potential $V_{p2}(r_2)$ will gain potential energy eV_{p2} and undergo a different change in wavelength to the first electron. Assuming they reached the specimen in phase (from a coherent source), this change in wavelength will result in a change in phase at the exit plane which is related to the difference in projected potential at that location. Thus the phase modulation of the exit plane wavefunction carries information about the structure (differences in projected potential) which we would like to image. Unfortunately if this exit plane wavefunction were to be measured using a CCD, only its intensity would be recorded and this phase modulation information would be lost, the phase modulation must be transferred into amplitude modulation in order to get any image contrast which can be measured, the process by which this

is achieved is referred to as phase contrast imaging.

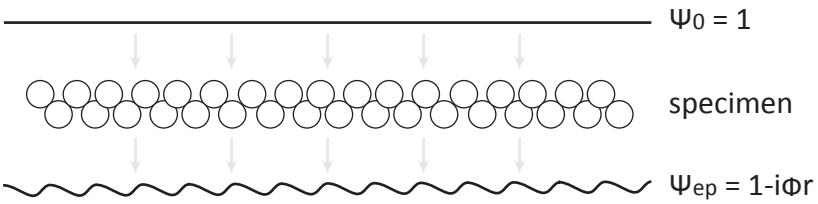

Figure 1.4: In the weak phase object approximation the phase of the incoming wavefront is modified by the electrostatic potential of the specimen resulting in the exit plane wave.

If the object can be considered a *weak* phase object ($\phi << 1$) then this phase modulation can be approximated by $\exp[-i\phi(r)] \approx 1 - i\phi(r)$ which is known as the weak phase object approximation (WPOA) [Williams and Carter 1996]. This allows us to express the exit plane wavefunction as the product of the incident wavefunction $\Psi_0 = 1$ and the phase modulation, giving $\Psi_{ep}(r) = 1 - i\phi(r)$ (see fig. 1.4). The electron wave at the back focal plane of objective lens is a Fourier transform of the complex exit plane wavefunction at the front focal plane of the lens. This is expressed in eq. (1.1) where k is the spatial frequency component conjugate variable to the spatial position r.

$$\Psi_{bfp}(\mathbf{k}) = FT\left[\Psi_{ep}(\mathbf{r})\right] = \Psi_{ep}(\mathbf{k}) \tag{1.1}$$

The electron wavefunction that is propagated to the image plane is not just the inverse Fourier transform of the exit plane wavefunction at the back focal plane however, this is because the objective lens is not an ideal lens, we have to take into account the transfer characteristics of the lens and there effect on the wavefunction propagation as shown in eq. (1.2) where $\otimes$ is used to denote a convolution.

$$\begin{aligned} \Psi_{image}(\mathbf{r}) &= FT^{-1}\left[\Psi_{ep}(\mathbf{k})t(\mathbf{k})\right] \\ &= \Psi_{ep}(\mathbf{r}) \otimes t(\mathbf{r}) \end{aligned} \tag{1.2}$$

Here $t(\mathbf{k})$ is the transfer function for the objective lens as a function of the spatial frequency, the form of this equation is given in detail in section 1.2.1.2. An ideal lens would have a constant transfer function at all spatial frequencies which would directly transfer the information from the exit wave into the image. In reality the objective lens suffers from many aberrations, these have the effect of creating an objective lens which causes a spatial frequency dependent phase shift to the

exit plane wavefunction, this spatial frequency dependent phase shift is actually responsible for generating image contrast from the phase modulation in the exit plane wave as shown in eq. (1.4).

By substituting $\Psi_{ep}(r) = 1 - i\phi(r)$ into eq. (1.2) for a weak phase object we arrive at

$$\Psi_{image}(\mathbf{r}) = 1 + \phi(\mathbf{r}) \otimes \Im[t(\mathbf{r})] - i\phi(\mathbf{r}) \otimes \Re[t(\mathbf{r})] \tag{1.3}$$

The recorded image is the intensity or square modulus of the electron wave at the image plane, for a weak phase object and neglecting second order terms this gives

$$\begin{aligned} I_{image}(\mathbf{r}) &= |\Psi_{image}(\mathbf{r})|^2 \\ &= 1 + 2(\phi(\mathbf{r}) \otimes \Im[t(\mathbf{r})]) \\ I_{image}(\mathbf{k}) &= FT[I_{image}(\mathbf{r})] \\ &= \delta(0) + 2\phi(\mathbf{k})\Im[t(\mathbf{k})] \end{aligned} \tag{1.4}$$

Equation (1.4) shows that the imaginary part of the lens transfer function $\Im[t(\mathbf{k})]$ is responsible for taking the information about the projected potential of the specimen which was stored in the phase of the of the exit plane wavefunction, and converting it into image contrast. In other words, the aberrations of the objective lens are necessary to form observable image contrast for weak phase objects that only affect the phase of the electron wavefunction, an ideal lens would produce no contrast.

1.2.1.2 Aberrations and their Importance

As mentioned previously, the transfer function of the objective lens controls the information that is transferred from the exit plane wavefunction into image contrast. The quality of the objective lens in the microscope has a crucial role on its transfer function and therefore on the formation of images in the microscope. The magnetic lenses that are used to focus electrons in a TEM cannot be manufactured with the same accuracy as the optical lenses used in light microscopy, this results in the presence of significant distortions to the images. An ideal lens would take the spherical wavefront coming from a point on the object and transform it into a spherical wavefront converging at a point in the image (fig. 1.5)). In a non ideal lens, if the deviation away from this spherical surface is δ, then this deviation can also be expressed as a phase shift $\chi = \frac{2\pi}{\lambda}\delta$. The aberration function for the lens $\chi(\boldsymbol{\alpha})$ describes this phase shift as a function of the angular deviation away from the optical axis $\boldsymbol{\alpha}$ and can be expressed as a power series with terms describing the

extent of the various different lens aberrations. Alternatively using $\alpha = k\lambda$ it can be expressed as a power series in terms of the spatial frequencies in the electron wave which describes how it will affect features that appear at a given frequency within the image [Erni 2010]. The overall transfer function for the lens $t(\mathbf{k})$ can be expressed in terms of this aberration function as in eq. (1.5).

$$t(\mathbf{k}) = \exp[-i\chi(\mathbf{k})] = \cos[\chi(\mathbf{k})] - i\sin[\chi(\mathbf{k})] \tag{1.5}$$

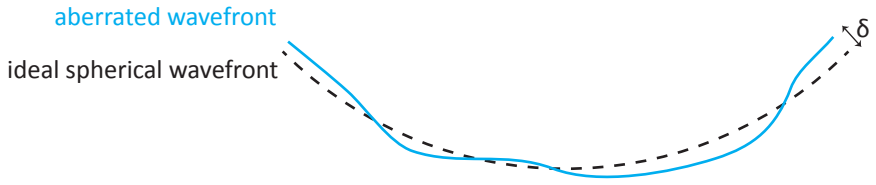

Figure 1.5: The aberration function is a measure of the displacement from an idealised spherical wavefront (displayed as a dashed line).

As only the imaginary part of the transfer function $\Im[t(\mathbf{k})]$ generates image contrast for weak phase objects, it is more common to refer to the phase contrast transfer function (PCTF) which is just the imaginary part of the transfer function.

$$PCTF(\mathbf{k}) = -\sin[\chi(\mathbf{k})] \tag{1.6}$$

If we only include the 2 primary symmetric aberrations, namely defocus C_1 and spherical aberration C_3 then we get eq. (1.7). This approximation is often used for none aberration corrected microscopes where the presence of a large fixed spherical aberration can often dominate the contributions from the other aberrations. The defocus is included as it is the main adjustable parameter that can be used to intentionally alter the lens transfer function to manually optimise the imaging conditions.

$$\chi(\mathbf{k}) = \frac{2\pi}{\lambda}\left[\frac{1}{2}(\mathbf{k}\lambda)^2 C_1 + \frac{1}{4}(\mathbf{k}\lambda)^4 C_3\right] \tag{1.7}$$

$$PCTF(\mathbf{k}) = -\sin\left(\frac{2\pi}{\lambda}\left[\frac{1}{2}(\mathbf{k}\lambda)^2 C_1 + \frac{1}{4}(\mathbf{k}\lambda)^4 C_3\right]\right) \tag{1.8}$$

If we switch to complex notation for the angle $\omega = \lambda k_x + i\lambda k_y$ and $\omega^* = \lambda k_x - i\lambda k_y$ then we can more conveniently rewrite the aberration function including

all axial aberrations up to the 5th order as eq. (1.9)[Erni 2010; Lentzen 2006].

$$\begin{aligned}\chi(\omega) =& \Re\left\{\frac{2\pi}{\lambda}\left[\frac{1}{2}\left(\omega\omega^*\right)C_1 + \frac{1}{2}\omega^{*2}A_1 + \frac{1}{3}\omega\omega^{*2}B_2 + \frac{1}{3}\omega^{*3}A_2 + \frac{1}{4}\omega\omega^{*3}S_3\right.\right.\\ &+ \frac{1}{4}\left(\omega\omega^*\right)^2 C_3 + \frac{1}{4}\omega^{*4}A_3 + \frac{1}{5}\omega^2\omega^{*3}B_4 + \frac{1}{5}\omega\omega^{*4}D_4 + \frac{1}{5}\omega^{*5}A_4 + \frac{1}{6}\left(\omega\omega^*\right)^3 C_5\\ &+ \left.\left.\frac{1}{6}\omega^2\omega^{*4}S_5 + \frac{1}{6}\omega\omega^{*5}R_5 + \frac{1}{6}\omega^{*6}A_5\right]\right\}\end{aligned} \tag{1.9}$$

In eq. (1.9), C_1 represents the defocus, C_3 the spherical aberration and C_5 the fifth-order spherical aberration, $A_{1,2,3,4}$ are 2,3,4,5-fold astigmatism respectively, and $B_{2,4}$ are 2,4-fold coma, $S_{3,5}$ are the 3rd, 5th order star aberration, and D_4 is the fifth order rosette aberration. The dependence of the aberrations upon ω means that electrons passing through the lens at different distances from the optic axis will encounter different phase shifts, or alternatively that different spatial frequencies in the image plane will have different phase shifts based on the size of the aberration coefficients. The phase shifts will also generally increase with increasing spatial frequency, or in the spatial domain, the aberrations will be far more important for the higher resolution information.

In phase contrast imaging the transfer function is what determines the contrast within the images based on the size of the various aberrations that are present. Whilst the defocus, 2-fold astigmatism and coma are easily adjustable within the microscope, the majority of the other aberrations are typically not. This means there is little control over of the transfer function and the higher resolution information will be distorted due to the presence of the higher order aberrations. In modern microscopes used for high resolution work, aberration correctors are installed between the lenses in the TEM, these allow for controlling the size of the aberrations in order to improve the lens transfer characteristics. Aberration correctors that are able to correct up to third order aberrations are now commonplace, but there are also newer generations of correctors able to correct for even higher order aberrations [Haider et al. 2009; Hosokawa et al. 2013]. All aberration correctors allow for tuning the amount of spherical aberration C_3 to a specific value, including negative values rather than just removing it altogether. This allows for both C_3 and C_1 to be used to design the transfer function optimally for imaging of a particular sample or producing the largest information passband possible. The ability to generate a negative coefficient of spherical aberration is also beneficial for imaging under certain circumstances [Urban et al. 2009], this is attributed to additive contributions from both amplitude and phase contrast under negative spherical aberration imaging conditions and can allow for imaging of lighter elements such as oxygen that cannot be resolved using

conventional positive spherical aberration imaging.

It is worth noting that if the values of all aberration coefficients were reduced to zero via aberration correction, the PCTF would be zero for all spatial frequencies, i.e. there would be no contrast in our images for a weak phase object. The presence of these aberrations is necessary to be able to image these kinds of samples without resorting to more elaborate schemes to generate contrast for changes in the phase of the exit plane wavefunction such as phase plates [Danev and Nagayama 2001; Nagayama and Danev 2008] or holography [Gabor 1948].

1.2.1.3 The Phase Contrast Transfer Function

The PCTF in eq. (1.7) is an oscillatory function, this means that for some spatial frequencies we will be generating positive contrast features when $PCTF > 0$ (i.e. atoms appear as white dots) but at other spatial frequencies will be generating negative contrast features $PCTF < 0$ (i.e. atoms appear as dark dots) this can make image interpretation very difficult. The ideal phase contrast transfer function would contain a wide passband that is approximately constant valued but there are often frequencies at which no information is transferred and oscillations at higher spatial frequencies making high resolution information hard to interpret even when it is present.

1.2.1.4 Point Resolution and Information Limit

The transfer of higher spatial frequencies is also complicated by additional factors, one such factor is the coherence of the electron source. The spatial coherence governs the relationship between the phase of electrons emitted from different positions on the source, and the temporal coherence governs the phase relationship between electrons emitted with different energies. Most electron sources are only partially coherent due to the fact they have finite sizes rather than being point sources and that they have small energy spreads rather than being monochromatic. This coherence is accounted for by the use of damping envelope functions which reduce the transfer of information at higher spatial frequencies [Wade and Frank 1977]. These coherence envelopes give rise to the concept of the information limit of the microscope, the maximum spatial frequency at which information is transferred to the image, in most microscopes it is the coherence of the electron source that determines the information limit, although for a highly monochromated source with high spatial coherence, the information limit could be determined by other factors [Uhlemann et al. 2013].

The envelopes for temporal and spatial coherence ($E_{temporal}(\mathbf{k})$ and $E_{spatial}(\mathbf{k})$ respectively) are given by [Erni 2010]

$$E_{temporal}(\mathbf{k}) = \exp[-\frac{1}{2}\lambda^2\pi^2\Delta^2\mathbf{k}^4] \tag{1.10}$$

$$E_{spatial}(\mathbf{k}) = \exp[-\beta^2\pi^2(\mathbf{k}^2C_1^2 + 2\lambda^2k^{\mathbf{4}}C_1C_3 + \lambda^4\mathbf{k}^6C_3^2)] \tag{1.11}$$

The temporal and spatial coherence envelopes are characterised by the defocus spread Δ and the semi-angle of convergence β respectively. These convergence envelopes lead to an effective transfer function $t(\mathbf{k})$ which can be written as a product of the objective lens transfer function and both envelope functions. We can also include the effects of an objective aperture to the transfer functions as a top-hat function $A(\mathbf{k})$ which removes all information transfer above a given spatial frequency which is determined by the size of the aperture. An example PCTF with the envelope functions displayed is given in fig. 1.6.

$$t(\mathbf{k}) = A(\mathbf{k}) \times E_{temporal}(\mathbf{k}) \times E_{spatial}(\mathbf{k}) \times \exp[-i\chi(\mathbf{k})] \tag{1.12}$$

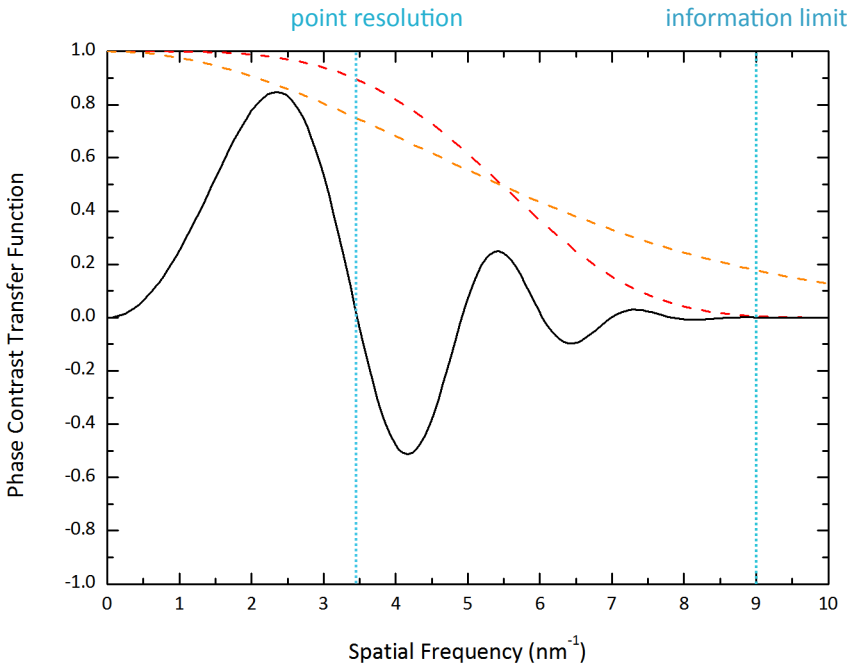

Figure 1.6: Example phase contrast transfer function for -20nm defocus and $1\mu m$ spherical aberration at 80kV. The envelopes due to temporal [red] and spatial coherence [orange] are plotted as dashed lines assuming a defocus spread of 3nm and a semi-angle of converge of 0.5mrad. The information limit due to the partial coherence and the point resolution are also indicated.

For high resolution work, an optimum choice of defocus exists for creating a wide passband in the PCTF to make the images as easy as possible to interpret, this is known as the Scherzer defocus [Scherzer 1949]. The wide passband means all spatial frequencies within this range will be transferred to the image with the same sign giving a direct correlation between image contrast and the actual object for information lower than this maximum spatial frequency. The size of this maximum spatial frequency is what determines the point resolution, the smallest size that is transferred to the image with a direct correlation to the object.

$$C_{1Scherzer} = -\sqrt{\frac{4}{3}\lambda C_3} \tag{1.13}$$

At the Scherzer defocus $C_{1Scherzer}$, the point resolution ρ_{sch} is determined by

$$\rho_{sch} \approx 0.66(\lambda^3 C_3)^{\frac{1}{4}} \tag{1.14}$$

1.3 Exit Wave Reconstruction

As has been described so far, the phase contrast method utilises the aberrations of the microscope to turn the phase variations in the exit plane wavefunction into image contrast which enables high resolution information about the sample to be to transferred to contrast within the images. This method is necessary for weak phase objects as the CCD camera itself cannot record variations of the phase in the wavefunction but the phase carries all of the information about our samples. Exit wave reconstruction is a technique which allows us to determine both the phase and the amplitude of the exit plane wavefunction by making multiple measurements of the intensity at several different image planes. These multiple measurements can be solved to determine the exit plane wavefunction. Once the full complex exit plane wavefunction has been determined, the effects of any residual aberrations within the determined exit plane wavefunction can also be computationally removed allowing for determination of an exit plane wavefunction that is not corrupted by any contrast transfer oscillations at all, this extends the point resolution in the reconstructed image up to the information limit of the microscope which can make the images much easier to interpret as well as giving direct access to the phase which is closely related to the projected atomic structure.

1.3.1 History of Phase Retrieval Methods

The possibility of using images at multiple focal planes to determine the phase of the exit wave was originally introduced by Schiske [Schiske 1968]. Since then numerous methods have been proposed as a means to determine the phase of the exit plane wave. This original idea was expanded to incorporate Wiener filtering as a means to optimally process noisy images [Schiske 1973], and numerous other iterative schemes using Gerchberg-Saxton type algorithms were also developed to retrieve the phase [Gerchberg 1972; Allen et al. 2004; Coene et al. 1996]. Another iterative scheme that included the non linear imaging effects was also developed by Kirkland [Kirkland 1984] as well methods based upon 3D Fourier transforms [Coene et al. 1992] and a method based upon the transport of intensity equation [Teague 1983]. An important aspect common to all of these methods is that the imaging conditions must be known accurately in order to recover the exit plane wavefunction.

The phase of the exit plane wavefunction can also be recovered in a TEM using electron holography. This method uses a bi-prism to split the incident electron beam and generate an additional reference wave which doesn't pass through the sample. When the wavefunction and reference wavefunction are brought together to generate an interference pattern the phase shift can be recovered from the interference pattern using Fourier transforms or iterative methods [Takeda et al. 1982; Fujita and

McCartney 2005].

Recent work on EWR has been focused on recovering the exit-wave from diffraction patterns using ptychography or Coherent Diffractive Imaging (CDI). CDI uses one or more diffraction patterns from a structure and attempts to solve the inverse scattering problem in order to determine the exit wave, usually via complex iterative solutions [DAlfonso et al. 2012; Morgan et al. 2013; D'Alfonso et al. 2014; Wang et al. 2013]. EWR has also been used to extract some 3D information from as little as one projection [Van Dyck and Chen 2012], and to improve the achievable image quality for radiation sensitive materials [Huang et al. 2014].

1.4 General Purpose Computing on Graphics Processing Units

The use of powerful computers is ubiquitous throughout modern scientific research. Many techniques require extensive computational resources for producing complex simulations or processing larger and larger volumes of data. Historically the performance of computers has been improving at a rate of around $2\times$ every 2 years [Moore 1965], however adapting algorithms to express large amounts of parallelism and developing them for GPUs can achieve anything up to a $10 - 100\times$ increase in performance while still using relatively inexpensive hardware without waiting ~ 10 years for microprocessors to advance in speed by such a factor. The current trend of increasing computational performance of a processor by adding additional computational units also decreases the likelihood of actually achieving such a performance increase solely through incremental hardware improvements without actually optimising algorithms to take advantage of parallel computing. In order to benefit from high performance modern hardware such as GPUs, software must be designed to take advantage of multiple parallel threads executing simultaneously. As an illustrative example of the relative performance of modern GPUs, in 1999 the worlds fastest supercomputer had a peak performance of 2.4 TFlops (2.4×10^{12} floating point operations per second), a modern GPU (AMD Radeon HD7950, launched Jan 2012) can be purchased today for under £300 that offers a higher peak performance of around 3.0 TFlops inside of a standard desktop personal computer.

Much of the power of GPUs is simply due to the fact that a modern GPU can consist of anything up to 3000 individual processors, and whilst each processor individually is not as powerful as a traditional modern CPU, great performance gains can be realized by performing numerous small calculations in parallel. GPUs also use their own memory rather than the host systems memory, this typically offers much higher bandwidth than exists between CPUs and host system memory which can increase performance in situations where memory bandwidth may be the bottleneck.

Many of the techniques mentioned previously in section 1.3.1 such as image registration and particularly iterative phase retrieval schemes can be computationally demanding and require substantial processing time in order to generate results. The increased pixel count and frame-rate of modern CCD cameras also has a huge impact on the size of datasets that are recorded using electron microscopes and a more efficient means of processing data that can lead to large time savings becomes increasingly valuable when processing larger datasets using these techniques.

1.4.1 History of General Purpose Computing on Graphics Processing Units

GPUs began appearing in computers in the early 90s and as the name suggests were used solely for rendering graphics faster than possible when utilising the CPU due to the specialisation of the hardware. These GPUs were designed to be able to perform large numbers of calculations simultaneously to be able to update every pixel on the screen numerous times a second, to do this they would contain multiple processor cores which could run in parallel, these cores would be much simpler and slower than a traditional CPU but for most computer graphics applications where each pixel could be determined in parallel, the overall performance would be much quicker than a CPU for this kind of specialised task. The term GPGPU was coined in 2002 and refers to General Purpose Computation on Graphics Processing Units, at this point in time it was only possible to perform computation on a GPU by effectively re-imagining the computational problem to look like the standard rendering pipeline used for drawing objects to the screen i.e. data would be stored as if it were a texture to be rendered, and the functions that operate on the data would be written as pixel shaders that would operate on the texture. Even with the inherent difficulties in programming for GPUs in the early stages, there were still tangible performance benefits that drove the GPGPU field towards its current position.

Now GPU computing is a relatively mature field, supported by the hardware manufacturers and computational performance is an advertised selling point of powerful GPUs, many GPUs are designed solely for this purpose and are unable to even drive a display device. GPUs can be programmed using syntax identical to standard programming languages like C or C++ and have many of the advanced capabilities that were previously only available to CPUs. Both major GPU hardware manufacturers have made significant efforts to improve the computational abilities of their products not just their performance for 3D applications like computer games and design product lines aimed purely at the GPU computing market. Almost any dedicated GPU bought today will offer support for some form of programming language designed to run general purpose computations on the graphics processor.

The use of GPU computing has also recently been seen across numerous

scientific fields including medical imaging (image registration), computational chemistry (molecular dynamics and density functional theory (DFT)) and even other aspects of microscopy such as tomography and bloch wave simulation [Shams et al. 2010b; Genovese et al. 2009; Götz et al. 2012; Pennington et al. 2014; Xu et al. 2010; Birk et al. 2014]. The use of GPUs is however restricted to problems which can be sufficiently divided in parallelizable tasks and so cannot be effectively applied to all problems which require large amounts of computation.

1.4.2 Graphics Processing Unit - Hardware Overview

In order to understand how to write code that runs efficiently on modern GPUs it is important to understand the underlying hardware and to explain some of the terminology that will be encountered throughout this thesis. A more detailed breakdown of the hardware inside a GPU and different memory types can be found in [Kirk and Wen-mei 2012]. The following section gives an overview of the general features of a GPU that facilitate its high performance for parallel calculations, however GPUs from different vendors and different generations of hardware are likely to all feature different designs. More specific information about nVidia hardware can be found in the nVidia CUDA C Programming Guide, and detailed information of AMD hardware can be found in the AMD Accelerated Parallel Processing OpenCL Programming Guide [NVIDIA Corporation 2014; Advanced Micro Devices Inc. 2013].

1.4.2.1 Cores + Multiprocessors

The computational units on a GPU are commonly referred to as cores, these are somewhat similar to a traditional CPU however they tend to be simpler in design than a CPU for a desktop computer. A modern GPU can have up to around 3000 cores, a modern CPU is likely to have only 4 cores. One of the major differences between a standard CPU and the cores on a GPU is that the cores on a GPU are effectively limited to performing the same actions at the same time as each other, whereas the 4 cores in a desktop computer CPU are perfectly capable of performing completely different actions independently of each other. This specialisation of the GPU is what enables it to offer such extreme performance at the cost of a lack of flexibility.

The individual cores in a GPU are grouped together into units called multiprocessors, a multiprocessor is a group of cores (often 32 cores) which also has its own individual resources such as a limited pool of memory that can only be accessed by the cores within the multiprocessor. When a function is run on a GPU, it is designed to operate on a large number of different threads which represent the number of times we would like the function to be called, i.e. one thread for every pixel within an image or every number in an array. These threads are grouped together into blocks

which have a maximum size of between 256-1536 threads depending on hardware limitations. Each block is scheduled to be computed on a single multiprocessor, when there are more blocks to be computed than available multiprocessors, some blocks will not be computed until other blocks have finished computing. The size of a block is usually much larger than the number of cores within a multiprocessor, so not every thread in the block is actually being computed simultaneously, the block is further subdivided into a *warp* (usually 32 threads) which is then computed simultaneously across the cores in the multiprocessor. If the multiprocessor would have to wait for some reason (i.e. waiting for a slow memory access) it can instantly swap to another warp within the block which is not waiting and work on that one for a while instead, this fast context switching is the main process that is used to amortize the cost of high latency accesses to GPU memory if there are enough warps available with enough arithmetic logic unit (ALU) operations to be performed.

The multiprocessor is designed in such a way that every core has to be performing the same type of computation at the same time to reduce the complexity of the design. This means conditional code such as an *if* statement which would lead to half of a warp trying to do something different to the other half can potentially cause problems, in this case the entire warp may have to calculate both possibilities of the *if* statement and discard the appropriate results. This can make code with numerous conditional branches less than optimal on GPUs although it is still supported. A well designed code should avoid warp divergence if at all possible so that every thread in a warp will follow the same branch of any conditional statements.

The multiprocessor also has finite memory resources, every number that has to be stored for each thread in a block takes up one or more registers, and on modern GPUs there can be around 32,000 registers per multiprocessor. A register is the fastest type of memory available but there is limited availability so some times other types of memory have to be used if the particular function to be called is sufficiently complex, the speed of accessing the various memory locations can be a key factor in determining the overall performance of a function.

This subdivision of work into blocks and threads allows for easy scalability to different types of hardware, the number of threads for a particular computation is usually much larger than the number of available cores, but as newer hardware is released and the number of cores is increased, the code will run faster without necessarily requiring any changes as the number of blocks executed simultaneously will be increased seamlessly. The code will also run on much less powerful hardware, albeit slower provided that the memory limit of the hardware is not exceeded. There is also a potential drawback that for work that cannot be split into a large enough number of threads, it may be impossible to make use of all the execution resources of the GPU and so the theoretical maximum performance will not be achieved. A

similar performance bottleneck can be encountered when a problem requires large numbers of memory accesses but not many arithmetic operations, in this case the multiprocessors may have nothing to work on whilst waiting for slow memory accesses, for this reason the ratio between memory accesses and arithmetic operations is an important factor when looking at the suitability of a problem for GPU computation.

1.4.2.2 Memory Types

GPUs have numerous different kinds of memory that should be used under different circumstances for optimum performance. The GPU has a single large pool of memory which is usually referred to as *global memory*, typically anywhere from 1GB to 6GB on modern hardware, enough to store very large arrays of data with millions of elements (1GB is enough storage for 2^{28} floating point numbers), this type of memory has a huge bandwidth (amount of data transferred per second) but also a very high latency (time for the memory access to be returned). Although the latency on accessing global memory is very large, part of this time can be amortized via the methods described previously. The global memory is typically used for storing any large amounts of data as the other memory resources are much smaller in comparison.

GPUs also have a type of memory referred to as *constant memory*, this is usually only around 64KB (typically enough for 16,384 numbers) and as the name suggests, cannot be modified by functions running on the GPU, it is effectively read-only. This can be used for storing small amounts of information that will be needed by lots of different threads as there is far less latency accessing constant memory than global memory.

Each multiprocessor has a small pool of memory called *local memory* or *shared memory*, this memory is accessible by every thread within the block that is running on the particular multiprocessor but not via different blocks on the same or other multiprocessors, it can also be modified during execution, this allows it to be used to write data from one thread and read it in another thread if communication is required between different threads. Shared memory is faster than global memory, but not as fast as registers, it also cannot be used to communicate between threads that aren't in the same block, this must be done using global memory but this is very slow, if lots of communication is required between threads then the problem may not see a substantial performance increase through GPU computing or may need to be significantly redesigned to conform to the GPU computation paradigm.

Local memory can also be used as a user managed cache, to prevent multiple reads and writes to the slower global memory or to allow multiple threads to cooperate on a single result that is modifiable by all of the threads. Synchronization techniques are available to ensure that every thread within a block reaches the same point in the calculation before continuing to ensure that a thread has written to shared memory

before a different thread tries to read the value, this is especially important when considering that only a single warp is in computation at once so some parts of a block will be at different stages in the computation without utilizing any synchronization.

It should be noted also that GPUs can only perform computations on data that has first been uploaded to the GPU memory. This adds an additional overhead when compared to performing calculations on the CPU as all data must be transferred to the GPU first and transferred back afterwards to view the results. Transferring data to the GPU from the host computers RAM has a much lower bandwidth than transferring memory on the GPU itself and can cause a bottleneck in the computation process. This also places an upper bound on the size of object that can be held in memory as a desktop computer can be fitted with much more RAM than is available on a GPU, the GPU can however cycle memory between itself and the host as it only needs to store what is relevant for the current computation. There is also the possibility of fitting a computer with multiple GPUs which can work on different parts of a problem simultaneously to offer further increased performance and greater available memory.

1.4.3 Graphics Processing Unit Computing Languages

Currently GPU computing is mostly done using either CUDA or OpenCL, these are 2 different programming languages designed to allow people to write code to be run on a GPU using a syntax similar to C/C++ and to provide all relevant functionality such as reading and writing to the GPU memory space. Programs written using CUDA or OpenCL can then be run on any GPU hardware that supports either CUDA or OpenCL respectively. OpenCL programs can additionally be run on CPUs if there is no GPU present which allows a single codebase to be developed and maintained to provide support for numerous different computational architectures. The 2 major GPU computing languages CUDA and OpenCL both sometimes use different terminology for the various types of memory and thread group structuring, but the programming languages are largely very similar to each other.

There are also other methods of providing support for GPU computing such as Direct Compute and C++ AMP.

1.4.3.1 CUDA

CUDA is the proprietary language by nVidia for GPU computing and is only compatible with nVidia hardware. CUDA has been available since 2007 and is compatible with all of nVidia's current range of hardware, nVidia also produce hardware based on there mainstream GPUs designed specifically for use with CUDA with increased performance. A typical program design with CUDA will proceed to operate serially on the host computers CPU until reaching a section that can be

efficiently implemented on a GPU, then relevant data will be uploaded to the GPU memory before the GPU processing can begin, and finally the results will be copied back from the GPU to the host.

Anything that is to be run on the GPU has to be written as a separate function which takes in data as arguments that must be uploaded to the GPU. The terminology used for a function that operates on a group of threads on the GPU is a *kernel*. Kernels are effectively written in standard C++ with a few additional keywords and each GPU thread runs the exact same kernel. In order to generate different results from the different threads whilst using the same kernel, each thread can determine its position relative to the other threads (which thread and which block etc.) and use this identification to determine which values to calculate and which memory locations to read from and write to. Many commonly used routines have already been written for CUDA and packed into libraries to speed up production of accelerated code, this includes functionality like Fourier transforms, and many linear algebra problems.

An example CUDA kernel is shown in kernel code 1 which takes two vectors of size *length* and calculates the results of adding each element from vector A to the same element from vector B and storing it in vector C. To ensure that every value of C is calculated the kernel would be called for a block size of $bsize <= 256$ and a number of blocks $nBlocks >= length/256$.

```
1  __global__ void cudaAddTwoVectors(float* A, float* B, float* C, int length)
2  {
3     // Determine which element to access based on which block
4     // and which thread within the block
5     int Index = threadIdx.x + blockIdx.x * blockDim.x;
6
7     // Don't do anything if Index is greater than vector length
8     if (Index >= length)
9        return ;
10
11    // This thread calculates one value of C=A+B
12    C[Index] = A[Index] + B[Index];
13 }
```

Kernel Code 1: Example CUDA kernel for addition of 2 vectors.

1.4.3.2 OpenCL

OpenCL is not restricted to running on just one manufacturers hardware, or even just restricted to GPUs in general. It can be run on nVidia hardware, AMD hardware, Intel CPUs, AMD CPUs, and even mobile hardware such as those in phones and tablets. Like CUDA the code that is intended to run in parallel has to be grouped into functions called kernels which get called for a large number of threads at once. Inside the kernels the thread is able to determine its own position within the group of threads and use this identification to ensure it calculates a different part of the

result to the other threads. OpenCL kernels are also written in C++. Like CUDA, OpenCL also has many common routines available as part of libraries including Fourier transforms and linear algebra.

An example OpenCL kernel is shown in kernel code 2 which takes two vectors of size *length* and calculates the results of adding each element from vector A to the same element from vector B and storing it in vector C. To ensure that every value of C is calculated the kernel would be called for a block size of $bsize <= 256$ and a number of blocks $nBlocks >= length/256$. The example is almost identical to the CUDA version in kernel code 1 aside from the location of the memory variables has to be declared explicitly, there is a built in function to determine thread position within the group instead of determining this manually in CUDA, and the whole kernel is prefaced by ___kernel rather than ___global___.

```
__kernel void openCLAddTwoVectors(__global float* A, __global float* B, __global float* C
   , int length)
{
   // Automatically determine the thread id based on block and thread number
   // for a given dimension (0).
   int Index = get_global_id(0);

   // Don't do anything if Index is greater than vector length
   if (Index >= length)
      return ;

   // This thread calculates one value of C=A+B
   C[Index] = A[Index] + B[Index];
}
```

Kernel Code 2: Example OpenCL kernel for addition of 2 vectors.

1.5 Thesis Outline

Chapter 2 gives an overview of the theory behind EWR as well as the theory behind the multislice approach to simulating phase contrast images. The theory behind determination of the aberration coefficients is also explained as it is an important part of the EWR procedure. Finally chapter 2 also covers the process behind the acquisition of suitable image series to be used for EWR.

Chapter 3 covers the development of a program designed for very fast and accurate simulations of phase contrast TEM images using the multislice approach. This includes the optimisation and future proofing of the multislice approach through the use of GPU computation.

Chapter 4 covers the development of a plugin for Digital Micrograph^{TM}designed to perform acquisition of focal series and processing of these series using EWR. Again GPU computing was used to significantly decrease the time taken to perform a full exit-wave reconstruction and to allow for the possibility of using more complex

procedures and improved image registration due to the increased computational power available.

In Chapter 5 the EWR plugin developed in Chapter 4 was used to acquire focal series and produce reconstructed phase images for different morphologies of self-assembled block copolymers.

In Chapter 6 the EWR plugin developed in Chapter 4 was used to acquire focal series and produce reconstructed phase images for both pristine and fluorinated graphene samples to determine the extent and precise atomic configuration of the fluorination. The multislice simulation software developed in Chapter 3 was also used to produce realistic simulated image series for several possible conformations of fluorinated graphene which were also reconstructed to verify the results produced from the experimental sample.

Chapter 7 summarises the conclusions drawn throughout this work and indicates possible directions of further research based upon this work.

Chapter 2

Methods

2.1 Introduction

This section aims to cover the theories behind the multislice technique which is used to simulate TEM images and the EWR algorithms that are used to determine the exit plane wavefunction from a focal series of images. Some other relevant techniques such as the determination of aberration coefficents and the experimental acquisition of focal series are also covered.

2.2 Multislice Image Simulation

The multislice theory was originally proposed in 1957 by Cowley and Moodie [Cowley and Moodie 1957] to describe the dynamically diffracted intensities from crystal structures, as this was well before computers became widely available this could not yet be used to produce simulated images. The method was adapted in 1974 [Goodman and Moodie 1974] into a numerical method that could be practically used to produce TEM image and diffraction pattern simulations from structure models. The next major advance was the inclusion of the fast Fourier transform (FFT) routine [Ishizuka and Uyeda 1977] which dramatically lowered the computational requirements for the calculation of convolutions at a time when computational power was not so abundant. Since then there have been many additional refinements to the original technique but the underlying principles and computational method remain the same. More recent work on the multislice procedure has focused on the ability for multislice procedures to produce accurate simulations at lower accelerating voltages where the high-energy approximation is less valid, this further increases the computational complexity of the method [Wacker and Schröder 2014; Cai and Chen 2012; Ming and Chen 2013].

The basic premise of the multislice technique is to divide the full structure we wish to simulate into thin slices in the direction of the electron beam, the effect of a

single thin slice on the electron wavefunction can be approximated by determining the transmission function of just the atoms within the thin slice using tabulated atomic scattering factors and multiplying the initial wavefunction by this transmission function. The wavefunction is then propagated over the thickness of the slice via Fresnel diffraction in a vacuum. The use of thin slices is a requirement to satisfy an assumption that the projected potential is small which is used to simplify the calculation.

The single slice technique can then be applied recursively by using as input for the second thin slice the wavefunction after the first thin slice to proceed towards the exit surface of the structure. Finally the effects of the objective lens transfer function can then be applied to the simulated exit plane wavefunction by choosing appropriate values for the aberration coefficients and convoluting the exit plane wavefunction with the real space objective lens transfer function to produce a simulated image.

2.2.1 The Wave Equation

The original derivation by Cowley and Moodie [Cowley and Moodie 1957] followed a physical optics approach but it has since been shown that the same results can be derived using operators starting from the Schrödinger equation for fast electrons travelling in the z-direction eq. (2.1) as shown in [Kirkland 2010]:

$$\frac{\partial \psi}{\partial z} = \left[\frac{i\lambda}{4\pi}\nabla_{xy}^2 + \frac{2mei\lambda}{4\pi\hbar^2}V(x,y,z)\right]\psi(x,y,z), \tag{2.1}$$

$$/ \tag{2.2}$$

where λ is the electron wavelength, m is the electron mass, e is the electron charge, and ψ is the electron wavefunction.

this can be written using operators as

$$\frac{\partial \psi}{\partial z} = [A+B]\psi(x,y,z) \tag{2.3}$$

$$A = \frac{i\lambda}{4\pi}\nabla_{xy}^2$$

$$B = i\sigma V(x,y,z)$$

where A and B are noncommuting operators and

$$\sigma = \frac{2\pi me\lambda}{h^2}$$

Equation (2.3) has solutions of the form

$$\psi(x,y,z) = \exp\left[\int_0^z [A(z') + B(z')]dz'\right]\psi(x,y,0) \tag{2.4}$$

If we offset the original value by z and integrate up to Δz

$$\psi(x,y,z+\Delta z) = \exp\left[\int_z^{z+\Delta z} [A(z') + B(z')]dz'\right]\psi(x,y,z) \tag{2.5}$$

This can be simplified to

$$\psi(x,y,z+\Delta z) = \exp\left[\Delta z \frac{i\lambda}{4\pi}\nabla_{xy}^2 + i\sigma v_{\Delta z}(x,y,z)\right]\psi(x,y,z) \tag{2.6}$$

Where $v_{\Delta z}(x,y,z)$ is the projected potential for a small slice of the specimen of thickness Δz between z and $z+\Delta z$.

$$v_{\Delta z}(x,y,z) = \int_z^{z+\Delta z} V(x,y,z')dz' \tag{2.7}$$

Solutions of the form $\exp(A\epsilon + B\epsilon)$ cannot be easily factorized if A and B are non commuting to give:

$$\exp(A\epsilon + B\epsilon) = \exp(A\epsilon)\exp(B\epsilon) + \frac{1}{2}[B,A]\epsilon^2 + \mathcal{O}(\epsilon^3) \tag{2.8}$$

Where $\mathcal{O}(\epsilon^3)$ represents a grouping of terms of of order 3 and above in ϵ Using eq. (2.8) with eq. (2.6) we can write the traditional multislice equation.

$$\psi(x,y,z+\Delta z) = \exp\left(\Delta z \frac{i\lambda}{4\pi}\nabla_{xy}^2\right) t(x,y,z)\psi(x,y,z) + \mathcal{O}(\Delta z^2) \tag{2.9}$$

Where the transmission function $t(x,y,z)$ for a given slice is given by:

$$t(x,y,z) = \exp\left[i\sigma v_{\Delta z}(x,y,z)\right] \tag{2.10}$$

The term $\exp\left(\Delta z \frac{i\lambda}{4\pi}\nabla_{xy}^2\right)$ can also be more conveniently expressed as [Kirkland 2010]:

$$\exp\left(\Delta z \frac{i\lambda}{4\pi}\nabla_{xy}^2\right) = p(x,y,\Delta z)\otimes \tag{2.11}$$

Where

$$p(x,y,\Delta z) = FT^{-1}\left[P(k,\Delta z)\right] \tag{2.12}$$

$$= FT^{-1}\left[\exp(-i\pi\lambda k^2 \Delta z)\right] \tag{2.13}$$

$$= \frac{1}{i\lambda\Delta z}\exp\left(\frac{i\pi}{\lambda\Delta z}(x^2+y^2)\right) \tag{2.14}$$

Here $p(x, y, \Delta z)$ is the propagator function which describes the evolution of the exit wave passing through vacuum over a distance of Δz. Together eq. (2.11) and eq. (2.9) give:

$$\psi(x, y, z + \Delta z) = p(x, y, \Delta z) \otimes [t(x, y, z)\psi(x, y, z)] + \mathcal{O}(\Delta z^2) \tag{2.15}$$

Alternatively this can be written in terms of the individual slices that make up the sample $n = 0, 1, 2...$ to make the recursive nature explicit:

$$\psi_{n+1}(x, y) = p(x, y, \Delta z_n) \otimes [t(x, y, z)\psi_n(x, y)] + \mathcal{O}(\Delta z^2) \tag{2.16}$$

Starting from an initial wavefunction $\psi_0(x, y)$, which for HRTEM is a plane wave representing a coherent electron beam spread evenly over the surface of the specimen, we can determine the exit plane wavefunction by successive application of eq. (2.16) over all the slices in the sample. Alternatively if we would like to simulate something like a convergent beam electron diffraction pattern, we could choose an initial wavefunction that represents the converged electron probe instead of a spread electron beam.

2.2.2 Specimen Slicing

In general the slices used in a multislice calculation need to be thin enough that each slice can be described as a weak phase object for some of the approximations to hold. It is generally also advantageous if the specimen can be described in terms of identical repeating layers or a repeating group of layers, this is purely from a computational standpoint as it allows the transmission function for a specific layer to be calculated once and then it can be reused for subsequent slices instead of being recalculated for each slice. Obviously if the structure to be simulated is not aligned with the optic axis of the microscope this will generally not be possible. In the case of structures with defects etc., the slices will not be repeating and so the transmission function must be calculated separately for each slice, this is important as calculation of the transmission functions is usually the most time consuming part of a multislice calculation.

The accuracy of the multislice calculation is heavily dependent upon the choice of slice thickness, in principle the error in a particular calculation should decrease as the slice thickness is lowered. The downside to choosing thinner slices is that we have to calculate more transmission functions due to the greater number of slices and so the result will take longer to generate. Unfortunately the approach of lowering the slice thickness to improve accuracy is not applicable for many multislice implementations; however an approximation that is often used is to replace the projected atomic potential for the atoms within a given slice by the total projected

potential for those atoms integrated between plus and minus infinity rather than just between the boundaries of the slice. This allows for a much easier analytical calculation of the projected potential, and each atom only has to feature in a single slice of the calculation, but is only really applicable for large slice thicknesses which decreases the accuracy of the calculation in general. This limitation is covered in more detail in section 3.2.1.

2.2.3 Exit Wave and Image Simulation

After the recursive formula in eq. (2.16) has been applied for every slice in the structure then the final resulting wavefunction is the exit plane wavefunction which carries information about the sample primarily in its phase. Whilst this itself is a valuable result, the exit plane wavefunction is not typically accessible in a microscope and so further steps are required to simulate a phase contrast image as would typically be viewed in the microscope. The transfer function for a specific set of imaging conditions (astigmatism, defocus and spherical aberration etc.) is calculated and then multiplied with the exit plane wavefunction in reciprocal space to give the image wavefunction eq. (1.2), the effects of partial coherence can also be taken into account by multiplying the transfer function by the damping envelopes given in eq. (1.10) and an aperture included as in eq. (1.12). The square modulus of the image wavefunction is then taken which is comparable with an image that would be recorded by a CCD camera. Further steps could be taken to incorporate the effects of imaging under a given dose of electrons, or with a specific detector, this is covered in section 3.2.4.

2.2.4 Available Software

There are numerous pieces of software available for performing simulations of TEM images from supplied structure files. Notable examples are computem (formerly TEMSIM) by Kirkland which is described in his book [Kirkland 2010], this software performs multislice calculations with and without partial coherence and is also capable of simulating diffraction patterns and CBED images as well as STEM images all using the multislice method. QSTEM is also a free alternative which is aimed primarily at simulation of STEM images but is also capable of simulating TEM images, this software is developed by Christoph T. Koch and has several enhancements over the standard multislice procedure and at improving the quality of the image simulations [Koch 2002]. Other popular pieces of software are JEMS by Pierre Stadelmann which can perform both multislice and Bloch-wave image simulations, and xHREM which is a commercialized package developed by HREM Research Inc although there are numerous other multislice implementations available [Stadelmann 1987].

The multislice software developed in this thesis is intended to explore the benefits that can be achieved within image simulation for electron microscopy by adopting GPU based computational methods to allow for a large increase in simulation complexity and also simulation speed, and to implement additional features which are not present in other software aimed at improving the accuracy of multislice calculations.

2.3 Exit Wave Reconstruction

Exit wave reconstruction can be performed in a number of different ways, but the typical method is called the focus variation method whereby a number of images are recorded at differing levels of focus to ensure information transfer at a broad range of spatial frequencies up to the information limit. Whilst it would be possible to vary other parameters than the defocus to the same effect it is usually far simpler to adjust the focus. There are also super-resolution schemes which aim to improve the resolution beyond the information limit of the microscope by employing a combination of different beam tilts in conjunction with different focus levels, but the principles between all methods are somewhat similar [Kirkland et al. 1995].

2.3.1 Exit Wave Reconstruction Algorithms

There are several different algorithms available in the literature for restoring the exit plane wavefunction from a given focal series of images each with their own benefits and difficulties. Some of the available methods are briefly outlined below.

The iterative wave function reconstruction method [Allen et al. 2004] initially assumes that the phase of the image wavefunction at every image plane is zero and the amplitude is therefore given by the square root of the recorded image intensity. These assumed image wavefunctions are then propagated to the exit plane using knowledge of the defocus under which the image was acquired and then averaged together. This average is taken to be an initial estimate of the exit plane wavefunction, it is then back propagated to each of the original image planes where the new back propagated phase is used as the next estimate of the phase at that image plane and the amplitude derived from the intensity as before. The process is then repeated iteratively until the exit plane wavefunction converges to the correct result. At each iteration the back propagated image waves resulting from the trial exit plane wavefunction can be compared against the input image series to test the ability of the exit plane wavefunction to reproduce the image series, and this figure of merit is used to determine when the exit plane wavefunction has converged to the correct answer.

The maximum likelihood formalism is another iterative method that utilises

a slightly different approach to iterative wavefunction restoration [Coene et al. 1996]. In this method an estimated exit plane wavefunction is used to simulate each image in the focal series using knowledge of the imaging conditions it was acquired with as before. The difference between these simulated images and the actual images is then minimized using either steepest descent or conjugate gradient or similar methods to drive the direction with which to update the exit plane wavefunction to minimise the error between the estimated and original image series. This method does not make assumptions about the linearity of the imaging process which are used in many other exit wave reconstruction schemes, this makes it more generally applicable but also computationally far more demanding than the other methods described here.

Another iterative method is the full resolution wave reconstruction method [Koch 2008], this method also incorporates non linear imaging effects as in the maximum likelihood approach, in this method the back propagated wavefunctions are all weighted by a spatial coherence term, and image alignment is incorporated into the iterative part of the reconstruction to be continually refined.

The exit plane wavefunction can also be determined using the transport of intensity equation [Hsieh et al. 2004; Ishizuka and Allman 2005]. This method relates the phase of the exit plane wavefunction to its intensity and the derivative of its intensity in the direction of wave propagation (i.e. as the focus is varied). This method also has the advantage of not making assumptions about the scattering object and so is just as valid for strong and weak phase objects.

Another type of exit wave reconstruction methods are those that make use of linear restoration filters to reproduce the exit plane wave. These consist of finding a set of filters r_i for each image c_i such that the exit plane wave can be restored via the relation

$$\psi'(k) = \sum_i r_i c_i(k) \tag{2.17}$$

These methods have one potential advantage over the iterative methods in that they require considerably less computation to produce a result. The paraboloid method [Op de Beeck et al. 1996] is one such method that uses linear restoration filters, the filter is chosen to only pick out the linear imaging components which are on a parabola in reciprocal space such that:

$$\psi'(k) = \sum_i c_i(k) \exp[-\pi\lambda k^2 df_i] \tag{2.18}$$

Where df_i is the defocus of image i. This method is sometimes used to produce a first approximate result relatively quickly, and then this result is used as the starting point for one of the more accurate iterative schemes such as maximum likelihood.

2.3.2 Focal and Tilt Series Reconstruction

The focal and tilt series reconstruction (FTSR) method uses a similar restoration filter approach but aims to improve upon the choice of filter by including the partial coherence of the electron source and presence of noise within the images into the formulation [Saxton and Smith 1985; Meyer et al. 2002; 2004]. This method is also not iterative and so the results can be computed relatively quickly in comparison with some of the other methods available. The FTSR method is the chosen method of EWR for the reconstructions within this thesis and so the theory behind this method is explained in further detail here.

If the image intensity of image i from a focal series is given by $I_i(x)$ then we can define the image contrast as

$$c_i(x) = \frac{I_i(x) - I_0}{I_0} \tag{2.19}$$

where I_0 is the background image intensity.

If we assume fully linear imaging then the image contrast can be related to the exit plane wave via the relation:

$$c_i(k) = \psi(k)\exp[-i\chi_i(\mathbf{k})] + \psi^*(-k)\exp[i\chi_i(-\mathbf{k})] + n_i \tag{2.20}$$

Where χ_i is the aberration function for image i (eq. (1.9)) and * is used to denote complex conjugation.

We then seek restoring filters that minimise the least squares difference between the determined wavefunction and experimental wavefunction in the presence of noise.

$$\left\langle |\sum_i r_i c_i - \psi|^2 \right\rangle \stackrel{!}{=} \min \tag{2.21}$$

Here the $< ... >$ represents an average over an ensemble of restorations characterised by an experimental wavefunction ψ with image noise n_i.

The solution to this problem is a restoration filter of the form [Meyer 2002]:

$$r_i(\mathbf{k}) = \frac{\left(\sum_i |\exp[-i\chi_i(-\mathbf{k})]|^2 + \nu\right)\exp[i\chi_i(\mathbf{k})] - (\sum_i \exp[-i\chi_i(\mathbf{k})]\exp[-i\chi_i(-\mathbf{k})])^*\exp[-i\chi_i(-\mathbf{k})]}{\left(\sum_i |\exp[-i\chi_i(-\mathbf{k})]|^2 + \nu\right)\left(\sum_i |\exp[-i\chi_i(\mathbf{k})]|^2 + \nu\right) - |\sum_i \exp[-i\chi_i(\mathbf{k})]\exp[-i\chi_i(-\mathbf{k})]|^2} \tag{2.22}$$

This filter when multiplied by the image contrast for a given image will preserve information in regions where the image transfer is expected to be high, and tend to zero in regions where the image transfer is expected to be low.

The restoration filters depend on the aberration function for the imaging conditions each image was taken under χ_i and the noise to signal ratio of the images ν.

The exit wave ψ' can be then be found using eq. (2.17) although to practically implement this there are some further complications that should also be mentioned. So far it has been assumed all the images c_i are images of the same object taken under different focal conditions, but it is almost impossible to achieve this practically as the acquisition of an image series will usually be affected by drift which means each successive image is of a slightly different area. Also, in order to calculate the aberration function for an image in the series as required for the filters, the values of the individual lens aberrations for each image must also be known and this is also typically not the case.

The problem with image drift can in principle be remedied easily by cropping an area from each image which corresponds to the exact same area on the sample, this can be achieved by several different methods outlined in section 2.3.2.1 and in more detail in section 4.3. The determination of the aberration coefficients is explained in section 2.5.

2.3.2.1 Image Registration

An important part of exit wave reconstruction is the ability to align all of the images in a focal series to a common origin, there are numerous possible image alignment methods available that are not necessarily specific to exit wave reconstruction. The accuracy of the registration process is critical to the performance of EWR and a necessary part of all reconstruction methods. For focal series registration, standard image registration procedures such as cross correlation are usually insufficient due to the significant changes in image contrast that can occur with changes in defocus. The techniques used to align the focal series within this thesis are covered in the software development section in section 4.3.

2.3.2.2 Defocus and Astigmatism Determination

When the defocus difference between all images in a focal series is known and all of the images have been aligned to a common origin, the electron wavefunction can then be determined at an arbitrary focal plane, this wavefunction will only be the exit plane wavefunction if we know the absolute focus level of each image not just the change in focus between images. Whilst there are techniques such as diffractogram matching to determine the focus level of an individual image in some circumstances

(see section 4.6.1.3), the absolute focus levels can also be found using the phase contrast index (PCI) approach [Meyer et al. 2002], this consists of calculating a trial reconstruction at an assumed focal plane, calculating a figure of merit f_{PCI} for a range of trial defocus values around the chosen focal plane using the trial reconstruction.

The phase contrast index function f_{PCI} is defined as

$$f_{PCI}(\mathbf{k}, C_1, A_1) = -\cos(\arg(\psi_i) + \arg(\psi_{i-}) + 2\gamma_s(\mathbf{k}, C_1, A_1)) \tag{2.23}$$

with

$$\gamma_s(\mathbf{k}, C_1, A_1) = \Re[\pi\lambda(\mathbf{k}\mathbf{k}^* C_1 + \mathbf{k}^{*2} A_1)] \tag{2.24}$$

and to simplify the equation complex notation is used for k such that

$$\mathbf{k} = k_x + ik_y, \mathbf{k}^* = k_x - ik_y \tag{2.25}$$

This function should be positive for for all values of $\mathbf{k}$ if the values of C_1 and A_1 used match the conditions in the reconstructed wavefunction. By trialling possible values for C_1 and A_1 around a predicted value and then integrating the f_{PCI} over all k we can find the best possible match for both defocus and astigmatism and then either repeat the reconstruction with the new values, or modify the original wavefunction accordingly to the exit plane.

2.3.2.3 Residual Aberration Correction

In addition to correcting the residual defocus and astigmatism present in the restored exit plane wavefunction, it is also possible to computationally correct for any of the higher order aberrations that may also be present due to having the full complex valued exit wave. It is not possible however to measure the extent of any of these aberrations from a focal series and reconstruction alone, the aberration coefficients must be determined separately. If the values of the higher order aberrations (such as C_3, A_2, B_2 etc.) are known, such as from an aberration corrected machine or otherwise (see section 2.5), they can be easily corrected for by applying another transfer function to the exit plane wavefunction which has the opposite values of the aberrations to be corrected.

2.3.2.4 Exit Wave Image Simulation

Just as it is possible to remove the presence of aberrations from an exit plane wavefunction, the aberrations can also be intentionally added in the same manner to produce a wavefunction at an arbitrary image plane with specific values for the other aberrations. By taking the square modulus of this wavefunction we can essentially simulate a standard TEM image from the determined exit plane wavefunction. This can be used to determine the effectiveness of a reconstruction by comparing the simulated focal series to the original focal series, this technique is used as part of some of the iterative EWR schemes to test the convergence of the solution.

2.3.3 Available Software

Many of the EWR methods described earlier have been implemented in software packages which are usually commercial or bundled with a microscopes operating software although some free software is also available. The iterative wave function reconstruction (IWFR) method and FTSR method have both been implemented as plugins for Digital Micrograph^{TM}by HREM Research Inc, these contain functionality to automatically acquire a focal series, align the images and perform the exit wave reconstruction process. The full resolution wavefunction reconstruction (FRWR) method of Koch has also been implemented as a plugin for Digital Micrograph^{TM}but this is currently only compatible with older versions of the software.

The maximum likelihood formalism has been implemented in TrueImage by FEI [Kübel and Thust 2006], this includes both the paraboloid method and maximum likelihood methods.

The exit wave reconstruction software developed as part of this thesis runs as a plugin under Digital Micrograph^{TM}, this allows it to integrate the image acquisition process by controlling the microscope as well as perform the reconstruction. This software was written to be as optimised and efficient as possible in order to determine the phase from a focal series in a very short time frame. It was also designed to be more flexible than some of the other software available by including different optional methods for the image alignment procedure, as well as the option to align extra image distortions as well as translation, this includes scale and rotation changes which were necessary additions for the work in chapter 6.

2.4 Focal Series Acquisition

A standard focal series for EWR is usually acquired with focus values evenly spaced across a sizeable range of defocus although this is not a requirement for the FTSR method (some methods strictly require evenly spaced focal planes). Images are usually taken underfocus at a range of defocus values around the Scherzer defocus

(eq. (1.13)) with a defocus step large enough to cause a shift in the abscissa of zero valued positions in the PCTF. It is possible to theoretically determine an optimum choice of defocus values at which to takes images [Buist et al. 1996b]. Whilst it is perfectly possible to acquire a suitable series of images manually, this job is usually performed using a script to control the acquisition via computer control, this can have numerous potential benefits besides being much easier for the microscope user. A script to perform focal series acquisition has been developed and incorporated into the EWR plugin(see section 4.2).

2.4.1 Computer Controlled Acquisition

The basic inputs to the focal series acquisition script are the number of images to be acquired, and the change in focus between each image within the series. As focal series are usually acquired somewhere around Scherzer defocus, the script has an input for the number of images to take underfocus from the starting position and an input for the number of images to be acquired overfocus and assumes that you are starting from approximately the Scherzer defocus position. The size of the focus step is input in nm as the defocus step calibration from the microscope is used to convert this into a change in the digital to analog converter (DAC) value for the lens (this value is calibrated as in section 2.5.3). The script moves to the most underfocus image plane first and then acquires all images in order underfocus to overfocus before returning to the starting focal position, an optional pause can be included between subsequent images if necessary.

The size of the focus step and number of images to use in a focal series is heavily dependent on the sample which is being imaged. In general, the results of a reconstruction will improve as the number of images in a series is increased, but one of the major downsides of EWR is the increased exposure to the electron beam from taking multiple images which can cause significant damage to the sample and thus may impose an upper limit on the number of images. The range of focus values used is designed to cause a significant change in the PCTF to ensure information transfer at all possible spatial frequencies.

A more thorough description of the image acquisition plugin used in given in section 4.2.

2.5 Aberration Determination

One of the major benefits of EWR is that it can computationally remove the effects of aberrations from a reconstructed exit plane wavefunction. This means that if all other factors were the same, an equally good reconstruction could be performed from an aberration corrected electron microscope as from a standard TEM as the

detrimental effects of the greater aberration coefficients in the standard TEM could be removed by the EWR process. In practice this is not the case as it is often simpler to perform EWR using an aberration corrected microscope as a the size of the aberration coefficients can be quantified by an aberration corrector.

2.5.1 Tilt Induced Aberrations

The importance of determining the values of the aberration coefficients for performing exit wave reconstruction necessitates an accurate method to be able to measure them. The defocus and astigmatism are typically the easiest aberrations to be measured owing to their significant impact on the shape of a defocussed diffractogram. The positions of the ring pattern in a diffractogram from an amorphous object is largely dependent on the defocus, and the shape of the rings is largely dependent on the astigmatism. With a suitable method to compare an experimental diffractogram to a series of simulated diffractograms the best matching astigmatism and defocus can be found relatively trivially.

The presence of other aberrations cannot be determined by looking at a diffractogram in a simple manner as they have no easily observable effect, instead the beam must be tilted which will cause an apparent change in defocus and astigmatism (which can be measured) which is dependent upon the values of the other aberrations. In order to determine the other aberration coefficients uniquely the beam must be tilted in several different directions or with several different magnitudes or a combination of both.

The aberration function is written again for convenience below:

$$\chi(\omega) = \Re\left\{\frac{2\pi}{\lambda}\left[\frac{1}{2}\left(\omega\omega^*\right)C_1 + \frac{1}{2}\omega^{*2}A_1 + \frac{1}{3}\omega\omega^{*2}B_2 + \frac{1}{3}\omega^{*3}A_2 + \frac{1}{4}\left(\omega\omega^*\right)^2 C_3\right]\right\} \tag{2.26}$$

The aberration function under a complex beam tilt of $\tau = t_x + it_y$ (where t_x and t_y are the tilts in orthogonal directions x and y) can be expressed as a power series expanded around the new origin (for small tilts) by [Erni 2010; Spence 2013]:

$$\begin{aligned}\chi'(\omega) &= \chi(\omega+\tau) - \chi(\tau) \\ &= \Re\left\{\frac{2\pi}{\lambda}\left[\frac{1}{2}\left(\omega\omega^*\right)C_1' + \frac{1}{2}\omega^{*2}A_1' + \frac{1}{3}\omega\omega^{*2}B_2' + \frac{1}{3}\omega^{*3}A_2' + \frac{1}{4}\left(\omega\omega^*\right)^2 C_3'\right]\right\}\end{aligned} \tag{2.27}$$

With the new effective aberration coefficients $C_1', A_1', B_2', A_2', C_3'$ given by

$$
\begin{aligned}
A_1' &= A_1 + 2A_2\tau^* + \frac{2}{3}B_2\tau + C_3\tau^2 \\
C_1' &= C_1 + \Re(\frac{4}{3}B_2\tau^*) + 2C_3\tau^*\tau \\
A_2' &= A_2 \\
B_2' &= B_2 + 3C_3\tau \\
C_3' &= C_3
\end{aligned}
$$

One method of determining the aberration coefficients from the apparent tilt induced defocus and astigmatism changes is the Zemlin tableau method.

2.5.2 The Zemlin Tableau Method

From eq. (2.27) we can see that the apparent value of the defocus and astigmatism under the influence of a beam tilt is affected to various degrees by the presence of the higher order aberrations. Because the apparent astigmatism is affected by a combination of all the other aberrations we need a method that is capable of determining the contribution from the individual aberration components. This method is commonly referred to as the Zemlin Tableau method [Zemlin et al. 1978; Saxton 1995; Uhlemann and Haider 1998], it involves using a number of different beam tilt magnitudes and directions and measuring the change in defocus and astigmatism at each tilt using diffractograms. If enough different beam tilts are used, then we have an overdetermined set of linear equations which we can solve using least squares techniques to find the individual aberration coefficients from measurements of just C_1' and A_1' and knowledge of the tilt magnitude and direction. A simulated Zemlin tableau is given in fig. 2.1.

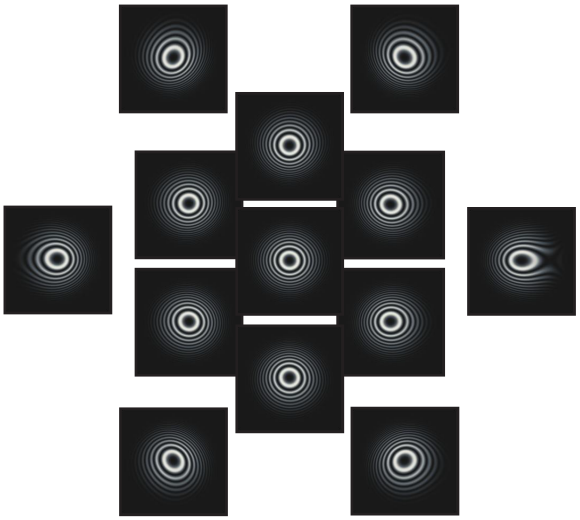

Figure 2.1: Simulated Zemlin tableau for aberrations $C_1 = -80nm$, $A_1 = 5 + 1i\,nm$, $C_3 = 7\mu m$, $A_2 = 45 - 10i\,nm$, $B_2 = 70 + 20i\,nm$, $A_3 = 6 + 1i\,\mu m$ and $S_3 = 4 + 2i\,\mu m$. The inner hexagon of diffractograms have a beam tilt magnitude of $18mrads$, and the outer hexagon have a beam tilt magnitude of $36mrads$, with the relative positions from the centre indicating the direction of the tilt.

2.5.3 Focal Step Calibration

To acquire a focal series with a specific value for the defocus change between each image in the focal series, it is necessary to know how much the aberration conditions change when the lens current is adjusted, rather than the absolute size of the aberrations via the previous method. For the defocus this can be determined relatively easily using diffractogram based methods as already outlined in section 2.5 to determine the value of the defocus at different lens strengths. By measuring the defocus C_{1a} at one lens strength, L_a, and then finding the amount of defocus C_{1b} at another lens strength , L_b, and dividing the change in defocus by the change in lens strength eq. (2.28) the calibration constant can be found assuming a linear relationship. This process can be repeated across numerous different ranges of defocus to get an average value or to verify the linearity of the relationship.

The lens strength is usually controlled from the computer via an integer value which is converted to changes in lens current by the DAC. An change of the lens strength by 1 unit on the DAC then represents the minimum possible focus change, df_{min}, that can be achieved via the objective lens. The change in defocus caused by adjusting the lens strength via this value will be given by $\Delta C_1 = df_{min} \times \Delta L$.

$$df_{min} = \frac{C_{1a} - C_{1b}}{L_a - L_b} \tag{2.28}$$

2.6 Chapter Summary

This chapter has summarised the theory behind the techniques used within thesis for both multislice image simulation and exit wave reconstruction. It also covers the theory behind the other techniques which are important in the practical application of EWR on a TEM.

Chapter 3

Multislice Simulation Software Development

3.1 Introduction

The complex contrast transfer mechanisms of a TEM make it difficult to interpret image contrast and relate it to the underlying structure being imaged. The ability to reliably reproduce the image intensities observed from a given microscope and structure via image simulations is often crucial to understanding the structure being observed in the microscope and also the conditions under which it is being observed.

Two main approaches exist for image simulation in electron microscopy, the Bloch-wave approach [Bethe 1928], and the multislice method [Cowley and Moodie 1957], which have each been adapted by numerous individuals to accommodate additional features or provide general improvements [Ishizuka and Uyeda 1977; Ming and Chen 2013; Croitoru et al. 2006; Van Dyck and Coene 1984]. The Bloch-wave approach is particularly suited to simulations from perfect crystal structures where the electron wavefunctions can be described by a set of Bloch states. When the structure cannot be described using a small unit cell (i.e. structures with defects, macromolecules etc.), the computational time required for the Bloch-wave approach is greatly increased and other approaches may be more practical. The multislice approach as described in detail in Section 2.2 divides the structure into numerous thin slices in the direction of the electron beam and then determines the wavefunction after propagation through one of these thin slices starting from an initial wavefunction. The approach is then repeated recursively over every slice in the sample to get the final result. This approach does not depend on the crystallinity of the structure being simulated (although it can be optimised for such cases) and is generally much faster for large structures than the Bloch-wave approach, for these reasons the development of software for TEM image simulation described in this thesis is based solely on the

multislice method.

3.2 Improving the Simulation Method

There are numerous adaptations to the original multislice method (hereby referred to as conventional multislice (CMS)) aimed at offering improved speed or accuracy. Early adaptations such as implementation of the FFT [Cooley and Tukey 1965] routine were primarily aimed at improving the computational speed of the algorithm, however the relative abundance of computational power that is now available has lead to an increased focus on increasing the accuracy and applicability of the algorithm rather than its speed. The use of FFTs changed the computational complexity of the calculation from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log_2 N)$, where N is the number of pixels, but little else has been done to improve the computational speed since then. Adaptations like the real space multislice (RSMS) method[Van Dyck and Coene 1984] can outperform CMS in some situations as it is only $\mathcal{O}(N)$, however the calculation for each pixel is typically longer than in CMS and there is also a restriction on the maximum usable slice thickness. Some authors of multislice simulation software have parallelised their software to various levels in order to provide increased speed on modern multi-core computers highlighting the importance of being able to deliver results under acceptable time constraints, however many popular image simulation packages remain unparallelised. The logical extension to parallelising software to work on a multi-core CPU is to parallelise the code to other highly parallel architectures such as GPUs which can offer significantly improved performance over CPUs, and also to optimise the algorithm to make maximum use of GPU functionality where this is possible.

Increasing the accuracy of the multislice algorithm is somewhat more difficult than improving its speed, however, many of the errors in the multislice simulation method arise from the approximations which are made in order to decrease the computational demands of the calculation. In removing these approximations we can hope to improve the accuracy of the calculation although this will increase the computational time concomitantly. By simultaneously improving the parallelisation and efficiency of the code at the same time as making it more accurate we can hope to make a version of multislice that is both faster and more accurate than the CMS which is still in widespread use.

One very simple approach to increase the accuracy of multislice calculations is to decrease the typical slice thickness used from around 1-2Å down to around 0.1Å, essentially using a much finer sampling in the beam direction, however this is ineffective for typical multislice implementations due to approximations made in calculating the atomic potentials described further in section 3.2.1. Removing this approximation should allow for the choice of arbitrary slice thicknesses where

accuracy or speed can be prioritised. There are also other approaches to the solution of the standard multislice equation eq. (2.1) which can lead to more accurate solutions, one such approach is the finite difference solution as described in section 3.2.3, but this is also not without its drawbacks. The multislice method has also been adapted to increase accuracy in alternative ways such as incorporating additional effects such as thermal diffuse scattering [Croitoru et al. 2006; Rosenauer et al. 2008], though the importance of this is mainly restricted to convergent beam electron diffraction (CBED) and STEM simulations and thus not discussed here.

In this chapter are presented further adaptations to the multislice routine aimed at both improving the accuracy of the simulation process in certain situations, and also additional incorporation of the effects of the noise and the CCD camera to the simulation. A significant emphasis is placed on greatly improving the speed of multislice calculations by adaptation of the multislice algorithm to highly parallel computational architectures such as GPUs and heavily optimising the resultant code. This has all been included together within the framework of a single software package aimed at producing simulations that can be tailored to match the precise experimental conditions of a given microscope and provide results rapidly even for very large and complex structures.

3.2.1 Atomic Potential Generation

Much of the accuracy of the multislice method is dependent on the choice of slice thickness (the method converges to a normalized result as the slice thickness tends towards zero [Anstis 1977]), however the CMS method usually involves an approximation aimed at improving the computational efficiency which prohibits the use of smaller slice thicknesses. The common approach when dividing the sample into thin slices is to choose a slice thickness of around 1-2Å, and then assume that all atoms that lie within a chosen slice lie in a horizontal plane at some height within the slice (usually the middle). This simplifies the calculation by allowing us to neglect the information about the actual z-position of the atoms within the slice, and in the case of aligned crystals where the slice thickness can be chosen to match exactly with the planar spacing of the crystal, will not introduce any error. This is problematic if the slice thickness cannot be chosen to match up with interplanar spacings within the structure to be simulated, or no such spacing exists like for irregular structures, in this case artefacts may be generated in the final image or diffraction pattern due to the incorrect handling of the z positions of the atoms.

If the chosen slice thickness is small enough, the difference between the actual z position, and the assumed z position would be negligibly small, however this cannot be achieved practically for two reasons. The simpler problem is that choosing a slice 10 times thinner will mean the calculation will take approximately 10 times

longer as there will be 10 times as many slices each requiring roughly the same amount of calculation as the previous thicker slices. The second problem is concerned with assumptions used in the calculation of the transmission function for the slice (eq. (2.7)).

$$t(x, y, z) = \exp\left[i\sigma v_{\Delta z}(x, y, z)\right] \tag{3.1}$$

Where

$$v_{\Delta z}(x, y, z) = \int_{z}^{z+\Delta z} V(x, y, z)dz \tag{3.2}$$

and

$$\sigma = \frac{2\pi me\lambda}{h^2} \tag{3.3}$$

Or for thick slices

$$v_{\Delta z}(x, y, z) \approx \int_{-\infty}^{+\infty} V(x, y, z)dz \tag{3.4}$$

The problem arises because the integral over the slice thickness in the projected potential in eq. (3.2) is typically not carried out. It is generally assumed that the entire projected potential of the atom is contained within the slice (which is a reasonable approximation for large slice thicknesses as the potential is far greater close to the atom than elsewhere). This allows for the full projected potential eq. (3.4) to be calculated analytically without performing the integral over the region of the slice via numerical methods which would require the generation of the potential at many different points for each atom at each position. This assumption breaks down for small slice thicknesses as the potential from around each atom should be spread over several slices not fully contained within any particular slice.

To accurately perform this integral as depicted in fig. 3.1 is far more computationally demanding as each atom will feature in the potential calculation for numerous slices (for thin slices the range of the potential from a single atom can span over numerous slices increasing the number of atoms included per slice by a factor proportional to the atomic potential cutoff distance divided by the slice thickness), and the integral has to be performed to a high accuracy using numerical methods which requires numerous potential calculations at various positions throughout the thickness of the slice. The removal of this assumption from the potential calculations, in addition to using the actual z-position of the atom rather than assuming it coincides with the slice centre allows for the choice of arbitrarily small slice thicknesses in the multislice procedure at the cost of a much more expensive calculation for the transmission function. The number of steps used in the numerical integration can be altered to provide additional fine control over the accuracy and computation time of the simulations. This approach has been taken in the development of the multislice software described in this chapter and together with the other improvements

presented in this chapter is referred to as the improved multislice (IMS) method.

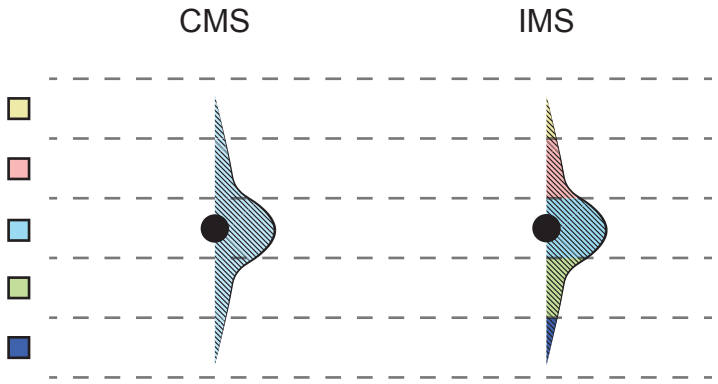

Figure 3.1: Schematic of the calculation of potentials due to a single atom in CMS and the IMS algorithm. The coloured regions represent the amount of the potential from the atom which is accounted for in each of the slices. In CMS the full potential is contained within the one slice the atom is located within, in IMS the potential is divided amongst several slices based on the range of the atomic potential.

3.2.2 Density Functional Theory

To further improve the way in which the transmission function is generated beyond the ways described in section 3.2.1 requires a more extensive treatment of the interactions between the individual atoms which are traditionally completely ignored. Typically the potential at a point is nothing more than a summation of the single atom potentials from all nearby atoms. Incorporation of additional effects such as bonding between atoms is far beyond the scope of this work, however calculations such as these are performed routinely using DFT for many kinds of systems. DFT offers a much more accurate calculation of the potential within a structure including many additional effects but is computationally far more demanding than the simplistic potential generation of multislice and as such usually limited to much smaller system sizes.

DFT calculations have already seen limited use for simulation of TEM images in situations where it is expected that the rudimentary approximations involved in the independent atom model would not accurately describe the electrostatic potential of the samples[Meyer et al. 2011; Kurasch et al. 2011]. Some popular DFT software packages are already able to output the potential sampled into 3D voxels without any modification to their code, this output can then be used within the multislice framework to simulate a TEM image without too much difficulty, by sampling the potential on a 3D grid of voxels the results can be simply translated to the multislice

arrangement of slices and pixels. In this case there is no requirement to calculate the projected potential as it can be taken directly from the DFT result and directly inserted into the transmission function of eq. (3.1). Care has to be taken to make sure the sampling of the DFT potential and multislice simulation are matched or that the results are otherwise appropriately interpolated, in the case of non rectangular unit cells, the DFT results may also not be calculated on a rectangular grid in which case interpolation is necessary to translate the results to be usable within multislice. This technique is still somewhat impractical for the simulation of larger structures due to the enormous increase in computational time required for DFT potential calculations in comparison with the independent atom model. Examples of images simulated from DFT derived potentials are given in fig. 3.2.

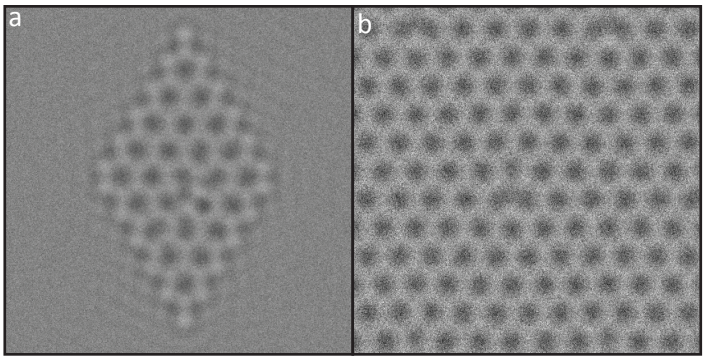

Figure 3.2: Simulations of nitrogen substitution in graphene sheet with a single vacancy based on a) independent atom model using DFT coordinates, and b) DFT derived potentials. The DFT potential has been interpolated and tiled onto a rectangular grid whereas the standard multislice simulation is just of an individual unit cell.

Figure 3.2 shows that the approach to calculating potentials in the multislice code still produces similar contrast to using DFT based potentials for some structures where the interactions between atoms do not have a dominant effect on the image contrast observed. In other cases however, there may be a significant difference between results simulated using the simplified model when compared to those based on DFT.

3.2.3 The Finite Difference Solution

As shown in section 2.2, the standard solution to the Schrödinger wave equation is formed by neglecting the second order differential part of the equation which leads to the introduction of a small error. It was also shown by Kirkland that the equation can be solved in an alternative method by Taylor expansion of the wavefunction ψ and then replacing its derivatives by their finite difference approximations. This

method is known as the finite difference method, with a Taylor expansion to first order we obtain the same solution as in Equation (2.15). If we choose to include higher order terms in the expansion and don't neglect the second order derivative term then we can calculate the result to a higher accuracy with a smaller error term.

Another advantageous feature of this approach is that we only need to calculate the potential at a given point, not the projected potential, this is potentially much faster than accurately integrating the potential over the slice as described in Section 3.2.1. The caveat to this method is that the solution is unstable unless extremely small slice thicknesses are used, and using extremely thin slices means the overall number of slices will be much larger and consequently it will take much longer. Ordinarily the requirement to use a much greater number of slices would make simulation via the finite difference method take a prohibitively long time, but the removal of the projected potential from the formulation in conjunction with the overall speed improvements made through parallelising the code (described in section 3.3) make it possible to perform a finite difference calculation in just a few minutes even for very large structures.

The finite difference solution as derived by Kirkland in [Kirkland 2010] is as follows:
First expand the wavefunction in orders of Δz up to third order.

$$\psi(z+\Delta z) = \psi(z) + \Delta z\frac{\partial\psi(z)}{\partial z} + \frac{1}{2!}\Delta z^2\frac{\partial^2\psi(z)}{\partial z^2} + \frac{1}{3!}\Delta z^3\frac{\partial^3\psi(z)}{\partial z^3} + \dots \tag{3.5}$$

$$\psi(z-\Delta z) = \psi(z) - \Delta z\frac{\partial\psi(z)}{\partial z} + \frac{1}{2!}\Delta z^2\frac{\partial^2\psi(z)}{\partial z^2} - \frac{1}{3!}\Delta z^3\frac{\partial^3\psi(z)}{\partial z^3} + \dots \tag{3.6}$$

Subtracting eq. (3.5) and eq. (3.6) gives the derivative:

$$\frac{\partial\psi(z)}{\partial z} = \frac{\psi(z+\Delta z) - \psi(z-\Delta z)}{2\Delta z} + \mathcal{O}(\Delta z^2) \tag{3.7}$$

Adding eq. (3.5) and eq. (3.6) gives the second derivative:

$$\frac{\partial^2\psi(z)}{\partial z^2} = \frac{\psi(z+\Delta z) - 2\psi(z) + \psi(z-\Delta z)}{\Delta z^2} + \mathcal{O}(\Delta z^2) \tag{3.8}$$

These finite difference approximations to the derivatives can then be inserted into the Schrödinger wave equation (Equation (2.1)). In contrast to the typical multislice formalism the second derivative term is maintained leading to the following equation.

$$\begin{aligned}\frac{\partial^2\psi(z)}{\partial z^2} =& \frac{\psi(z+\Delta z)-2\psi(z)+\psi(z-\Delta z)}{\Delta z^2} \\ &+ \frac{4\pi i}{\lambda}\frac{\psi(z+\Delta z)-\psi(z-\Delta z)}{2\Delta z} \\ &+ \nabla_{xy}^2\psi(z) + \frac{8\pi^2 me}{h^2}V(x,y,z)\psi(z) + \mathcal{O}(\Delta z^2) = 0\end{aligned} \tag{3.9}$$

After rearranging and multiplication by Δz^2 this gives Equation (3.10).

$$\begin{aligned}\psi(x,y,z+\Delta z) =& \frac{1}{1+\frac{2\pi i\Delta z}{h}}\left[2-\Delta z^2\left(\nabla_{xy}^2+\frac{8\pi^2 me}{h^2}V(x,y,z)\right)\right]\psi(x,y,z) \\ &- \frac{1-\frac{2\pi i\Delta z}{h}}{1+\frac{2\pi i\Delta z}{h}}\psi(x,y,z-\Delta z) + \mathcal{O}\left(\frac{\Delta z^4}{1+\frac{2\pi i\Delta z}{h}}\right)\end{aligned} \tag{3.10}$$

The error in eq. (3.10) is of a higher order in Δz than eq. (2.15) and we haven't neglected the second derivative as was done for traditional multislice. This method is a 3 plane slice method whereas traditional multislice is a 2 plane slice method[1]. The tricky ∇_{xy}^2 term can be more easily dealt with via a Fourier Transform.

$$\nabla_{xy}^2\psi(x,y) = FFT^{-1}\left[-4\pi^2k^2\cdot FFT\left[\psi\left(x,y\right)\right]\right] \tag{3.11}$$

Based on eq. (3.11) alone this method has roughly the same computational requirements as traditional multislice, requiring slightly more memory to store the extra wavefunction at each slice and performing some additional Fourier transforms per slice. The increased computational demand of the 3 slice finite difference method comes primarily from the requirement to use much thinner slices, the potential calculation at each slice is actually simpler due to the lack of the integral over the potential. It can be shown that the traditional multislice method is unconditionally stable regardless of slice thickness [Kirkland 2010], a slice thickness of typically 1-2Å is used as a compromise between accuracy and computational speed, when the slice thickness is smaller artefacts are introduced due to the incorrect projected potential assumption (section 3.2.1). The finite difference method however is only stable under the following criterion:

$$k_e^2 < \frac{1}{4\pi^2\Delta z^2}\left[\sqrt{1+4\pi^2\Delta z^2/\lambda^2}-2\right] + \frac{2\pi meV}{h^2} \tag{3.12}$$

Here k_e is the maximum frequency in the simulation (bandwidth limit) and this is coupled with the slice thickness Δz. Either the slice thickness or the bandwidth

[1] the wavefunction at $z+\Delta z$ is calculated from knowledge of the wavefunction at both z and $z-\Delta z$, as opposed to just the wavefunction at z

limited maximum frequency must be reduced to ensure a stable solution. Typically this results in a slice thickness of the order 0.1Å or less resulting in 1 to 2 orders of magnitude more slices which can lead to a calculation time which is also approximately 1 to 2 orders of magnitude longer. The much smaller slice thicknesses, along with the inherently more accurate formalism should make the results more accurate than the CMS method however.

3.2.4 Simulating the Effects of CCD Cameras and Electron Dose

Another important aspect of simulation of TEM images is the ability to quantitatively compare between simulated images and experimental images. Typical simulations take no account of the detector characteristics, which often have a marked effect on the final image, noise can also be added to images without regard for the actual noise sources and noise characteristics and dependence on parameters such as the electron dose. This work also aims to incorporate the detector response into the simulation process to allow for more realistic image simulations. The most important figure of merit used to describe the performance of different electron detectors is the detective quantum efficiency (DQE)[Herrmann and Krahl 1982], this along with the noise power spectrum (NPS) and modulation transfer function (MTF) describe the spatial frequency dependent noise and signal transfer of an electron detector, or how its performance varies based on the spatial frequency of the image features in question [Ruskin et al. 2013]. Typically a detector's performance will lower considerably for high frequency information and so taking an image at a higher magnification will move this content to a lower frequency on the detector where it is imaged better at the expense of a minimized field of view. Increasing magnification without adjusting the electron dose per area will also lower the number of electrons per pixel on the detector which will reduce the contrast. A better electron detector (direct detection camera for example) will have a higher DQE for a given spatial frequency which can be included within the simulation process.

The DQE is defined as the ratio between the squares of the output and input signal to noise ratio (SNR) of a detector as a function of spatial frequency [Herrmann and Krahl 1982]. For a perfect detection system this would be unity across the entire frequency range, in practice there is a sharp descent in DQE at higher spatial frequencies. Some typical DQEs for a range of detection systems are shown in fig. 3.3 as well as the measured DQE for a Gatan Orius 1000 at 80kV in fig. 3.4, this is determined using the MTF and NPS as in eq. (3.14), the MTF and NPS were both measured using a routine within the FTSR software package by HREM Research

Inc on an Jeol ARM 200-F at Warwick.

$$DQE(k) = \frac{SNR^2_{out}(k)}{SNR^2_{in}(k)} \tag{3.13}$$

This can also be expressed in terms of the MTF and normalized noise power spectrum (NNPS) or noise transfer function (NTF).

$$\begin{aligned} DQE(k) =& \frac{MTF^2(k)}{NNPS(k)} \\ =& \frac{MTF^2(k)}{NTF^2(k)} \end{aligned} \tag{3.14}$$

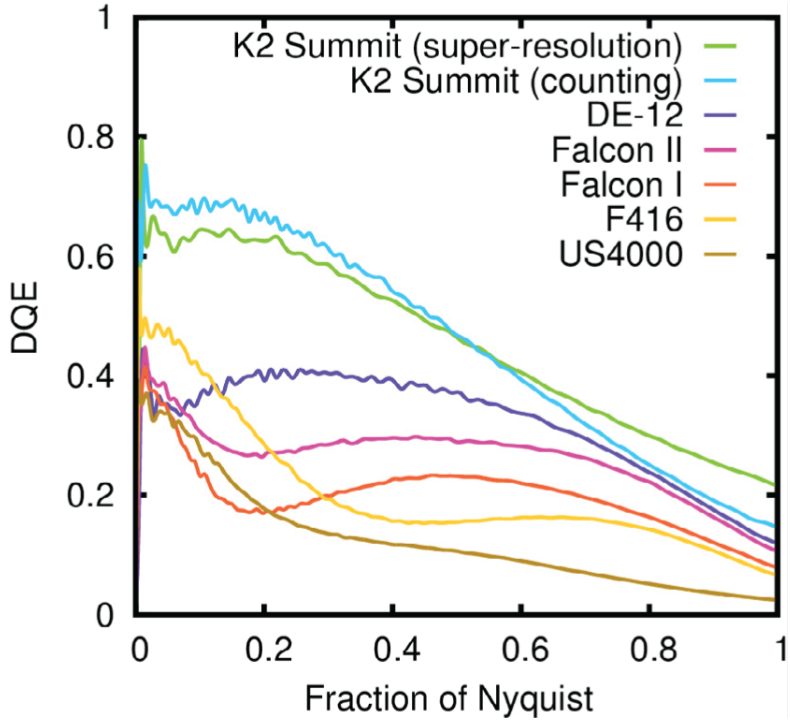

Figure 3.3: detective quantum efficiencies for various available electron detectors including CCD's and direct electron detectors. Figure taken from [Ruskin et al. 2013].

The CCD in an electron microscope can be characterized using relatively simple experiments to measure the MTF and NPS [Nakashima and Johnson 2003; Ruskin et al. 2013; Lubk et al. 2012]. Including CCD performance as part of the simulation allows for simulations which more accurately represent the conditions of the microscope in question. Because the noise and signal are transferred differently within CCD camera their effects must be included within the simulation routine

itself as opposed to simply adding noise over the perfect simulation result at the end.

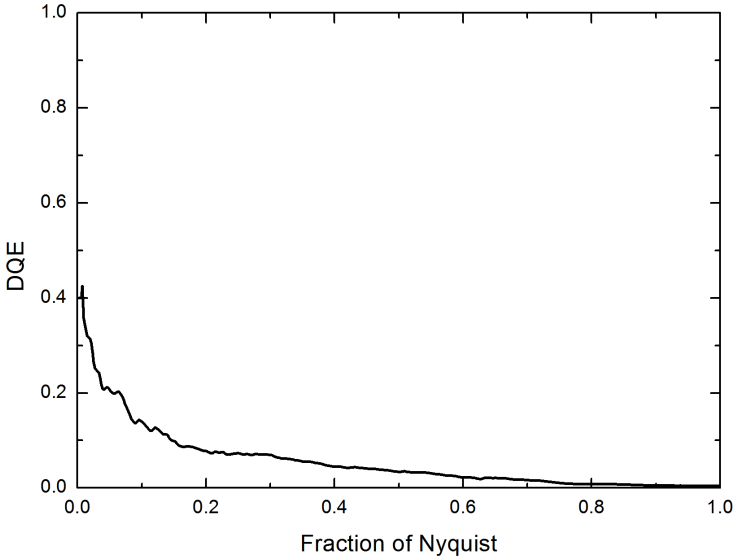

Figure 3.4: Experimentally determined detective quantum efficiency for Gatan Orius 1000 charge-coupled device Camera at 80kV accelerating voltage on a Jeol ARM200-F at Warwick.

The simulation of the detection process including CCD characteristics can be summarized into 4 steps [Vulović et al. 2013]. The first step is to damp the perfect simulated signal by the ratio between signal and noise transfer. This signal is then multiplied by the electron flux expected per pixel Φ_e, this is determined based on the electron dose for which the image is being simulated. The value at each pixel should now be representative of the mean expected number of electrons, shot noise contributions are incorporated by replacing the value at each pixel c_i by a random value taken from a Poisson distribution with mean c_i. This signal is then further damped by the NTF and then multiplied by the CCD conversion factor CF, this converts the value from a number of electrons to analog-to-digital units (ADUs) representative of the values obtained from the specific CCD. Finally the signal is modified by adding readout noise I_{rn} and dark current I_{dc}. If the image size calibration of a specific microscope is known for a specific magnification this can also be used within the simulation process. In this way an image can be simulated at the same magnification, electron dose and exposure time, with the same number of pixels as would be used for image acquisition in an electron microscope and these simulated images should be quantitatively comparable with the experimental images. This method of incorporating the electron dose and CCD characteristics make it possible to simulate effects such as the difference in acquiring images with CCD

binning of various levels or at specific magnifications which is not possible with most simulation routines.

$$I(x,y) = \\ I_{rn} + I_{dc} + CF \cdot FFT^{-1}\left[FFT\left[Poiss\left(\Phi_e \cdot FFT^{-1}\left[I_0(k)\sqrt{DQE(k)}\right]\right)\right] \cdot NTF(k)\right] \tag{3.15}$$

3.3 Parallelisation

The multislice algorithm is one that is particularly well suited to being parallelised, many of the operations that are performed over every pixel within an image can be performed independently with no dependence on the results of the other pixels, this means that it is possible to calculate each pixel simultaneously given enough computational resources. The multislice algorithm is also particularly well suited to GPU parallelisation in particular, the architecture of GPUs are well suited to tasks that can be divided into many thousands of smaller tasks, unlike a multi-core CPU which is optimised for working on a much smaller number of simultaneous tasks. In the case of multislice simulation, we are usually dealing with millions of pixels per simulated image, each of which can for the most part be calculated independently from one another. This large workload is enough to effectively divide over the execution units of a GPU to allow it to achieve high performance and utilisation.

Figure 3.5 and fig. 3.6 show an overview of the different elements present in a GPU. The GPU is logically subdivided into several multiprocessors which each have their own private memory caches, as well as access to the global memory cache shared by all multiprocessors. These private resources can be accessed at low latency with little contention but the global memory accesses will suffer much greater delay so it is advantageous to make use of them where possible. The large number of individual cores are what make it possible to calculate the results of multiple individual calculations simultaneously.

The maximum performance improvements are gained from GPU parallelisation when we are performing the same operations at the same time for every pixel in the data set (e.g. multiplying every pixel in an image with the same pixel from a different image), and as will be shown, the multislice algorithm can with a few exceptions, be optimised to fit this model very well by taking advantage of local memory on GPUs.

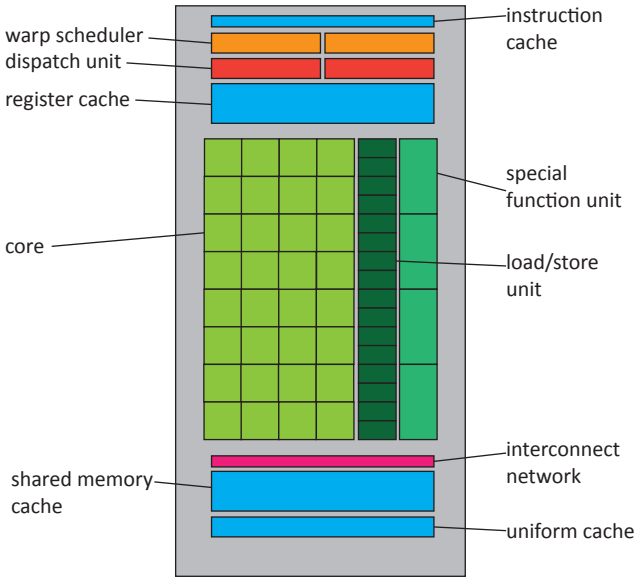

Figure 3.5: A schematic of the design for a single multiprocessor in a modern GPU. It contains numerous individual processors, a local memory cache, and specialised hardware for the calculation of transcendental functions.

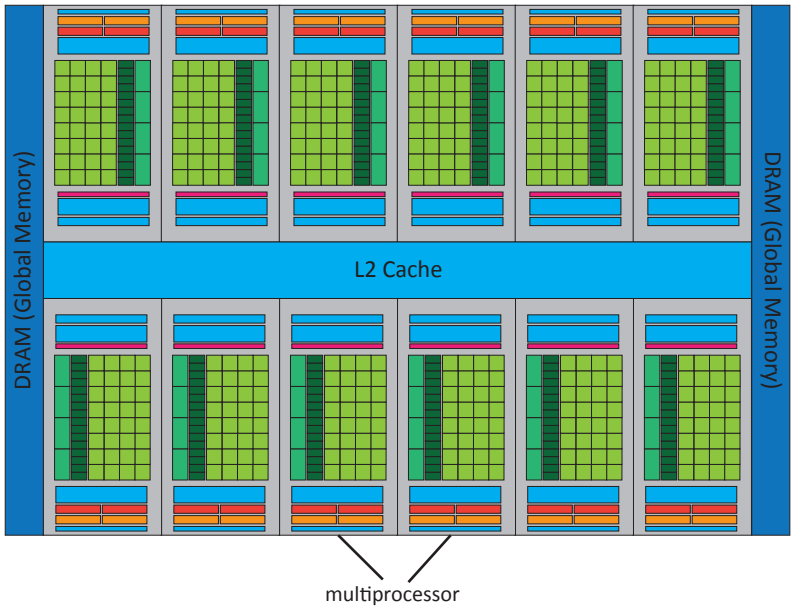

Figure 3.6: A schematic overview of the design of a modern GPU. The GPU is composed of numerous multiprocessors and a large shared memory pool, the design of a single multiprocessor is shown in fig. 3.5.

3.3.1 Optimisation for Graphics Processing Unit Architecture

As explained in section 1.4.3, the architecture of a GPU is substantially different from that of a traditional CPU, and as such, in order to extract the best performance many programs have to be completely redesigned rather than just being simply directly translated. An simplified way of looking at the problem is that each of the many individual processors on the GPU will be calculating the result for an individual pixel at the same time as all the other processors until all the pixels have been calculated. However, these processors are also grouped together into small clusters of processors which have some shared resources. These small clusters of processors are designed in such a way that they all have to operate in unison with each other and cannot be performing different tasks. Consequently the optimum performance can be obtained when all processors within a cluster are performing the same 'action' and struggle when there is a lot of conditional branching within the code. There are also performance benefits when processors within a cluster are accessing the same memory location or contiguous memory locations which can then be grouped into single transactions. GPUs also have different types of memory available to them, rather than just a large pool of random access memory (RAM), some of these memory types have very limited sizes but offer much faster access, others can also be seen by processors in the same cluster, and some memory is specific

to the individual processor in question. In many cases it is the management of these memory accesses that can have the most significant impact on the performance of a program.

Much of the optimisation of the multislice code is concerned primarily with calculation of the transmission function (see section 2.2.1), for every slice in the calculation another transmission function must be calculated, this includes calculating the contribution to the potential at each pixel from each atom in the sample which can obviously be very time consuming. If we include the contribution to the potential from every single atom in the sample at every pixel and every slice, this would take a prohibitively long time, so we first seek ways to simplify the problem without adversely affecting the result. The atomic potential has an approximate $1/r$ dependence which makes its influence rather small past a certain distance, for this reason we choose to discard contributions from atoms further away than a predefined cutoff radius. This limits the number of atoms to be included in the calculation per pixel to a much more amenable value with little effect on the results. If more accuracy is required the cutoff value can be freely modified. Additionally, if a processor cluster on the GPU is tasked with calculating the potential for pixels that are all in immediate proximity with each other, then we can also write the code in such a way that every pixel in each cluster only sees the potential contributions from exactly the same set of atoms. In this way we can make used of the shared resources of a processor cluster to enable more efficient access to the data, and ensure there is no divergence within the workgroup. This lowers the required memory bandwidth significantly because only one memory access is required to serve every single processor within the cluster and the lack of code branching ensures the code is efficiently executed.

This method of parallelisation for the transmission function calculation is different from the one adopted by Kirkland in his TEMSIM program [Kirkland 2010]. Kirkland chooses to parallelise over the set of atoms that feature in the slice, and then only update a small range of pixels around the position of each atom. This may be slightly more efficient when there are considerable amounts of the simulated area within a given slice that doesn't feature any atoms, but it is not the best approach to take for GPU parallelisation, at least not without significant modifications. This is because the limited number of atoms in a slice would not be enough to make use of all the resources available on a GPU. Parallel execution can also become troublesome when numerous processors are trying to write data back to the same location (i.e. if two processors were trying to update values at the same pixel at the same time), this is avoided by having the individual processors calculate separate pixels so they are all writing to distinct locations.

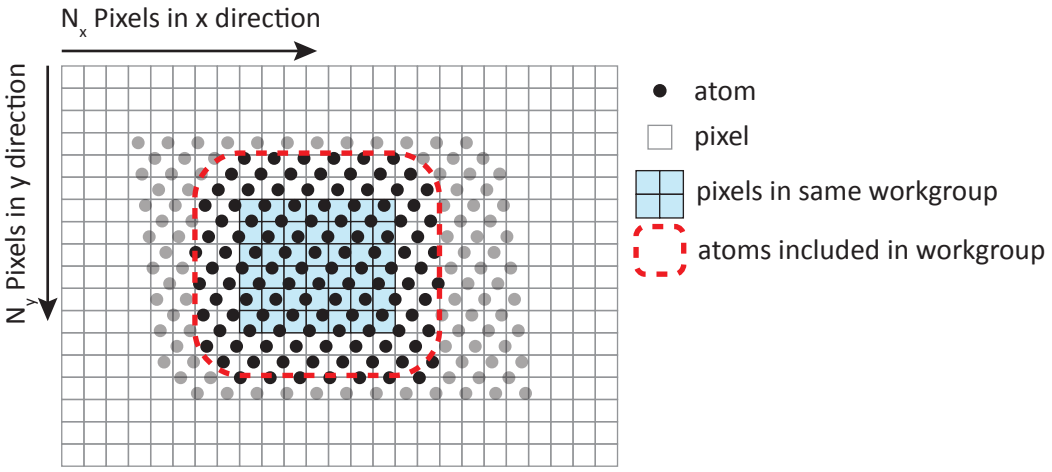

Figure 3.7: Schematic of the subdivision of the transmission function calculation to allow for efficient implementation on a GPU. All processors in a workgroup operate on pixels in immediate proximity with each other, this allows them to all use the same subset of atoms improving the calculation efficiency.

The potential calculation from a large subset of atoms is further optimised by pre-sorting all of the atoms into groups of atoms which share similar spatial co-ordinates (atoms that are close together). The atoms are then stored in memory in order of the group they were assigned to. Each processor cluster then only needs to know which subgroups of atoms it should load, which is in turn based on the spatial location to which it was assigned as depicted in fig. 3.7. This is further reduced to requiring just the starting memory location for each of these subgroups. This is more efficient than searching through all the atoms in the slice when only some are in close proximity. All processors within a workgroup will load the exact same atom groups to make sure there is no divergence within the workgroup. This system provides excellent performance and good performance scaling with increasing system size. This method also ensures that if a region of the simulation has fewer atoms, that its execution will complete faster freeing up these resources to work on another region of the simulation.

One downside to this approach, is that the coordinates for the atoms are stored within the *global* memory of the GPU due to size constraints. A straightforward implementation of the GPU kernel would have each multiprocessor loop through the subset of atoms in tandem with each other but this will provide poor performance on a GPU due to the memory access pattern. If each processor within a multiprocessor is trying to access data about the same atom at the same time, these memory accesses will be serialized, so each processor will have to wait for the others to get access

to the data. The optimum memory access model is when each processor wants to access the data element contiguous to the one from its neighbouring processor, this allows a large block of memory to be retrieved in a single memory access which is much faster.

In order to make this kernel more efficient by utilising this improved memory access model, an additional step can be implemented in which the atom coordinates are fetched from the *global* memory and stored in *local* memory in a coalesced access pattern before the potentials are to be calculated. The coordinates can then be accessed from local memory just as before, however local memory can offer broadcasted memory reads where it is efficient for every processor to access the same memory element at once. Whilst this adds more complexity to the GPU kernel as atoms are read initially from global memory and then subsequently from local memory, the improved memory access efficiency is enough to increase the overall speed of the program. Without this additional step each thread would have to load each atom individually rather than each thread being responsible for loading a single atom on behalf of the workgroup. This modification lowers the total number of reads operations from the high latency global memory by a large factor as atoms will be retrieved in groups of 16 rather than individually and each atom will also be loaded from global memory just once per workgroup rather than once per thread. This pattern also greatly reduces the possibility of having serialized memory accesses as they can now only occur when different workgroups try to access the same location in global memory which is now significantly less frequent.

The actual calculation of the potential contribution from an individual atom is based upon a parametrisation of the potential into 12 variables for each atom type ($a_{iz}, b_{iz}, c_{iz}, d_{iz}$ for $i = 1$ to 3 where z is atomic number) as used in eq. (3.16) [Kirkland 2010]. There are several such parametrisations in the literature [Weickenmeier and Kohl 1991; Doyle and Turner 1968; Rez et al. 1994] which can be used for calculation of single atom potentials. In many traditional multislice implementations, the potential would be pre-calculated at large number of different radii, and then these values would be recalled from a lookup table (LUT) instead of recalculated each time they are required, with a CPU based program it can often be faster to recall a number from memory than to perform the reasonably complex calculation to get the potential for a given atomic number and radius. For GPUs the situation is not as straightforward, performing large numbers of relatively simple calculations does not take much time, but recalling values from memory is quite an expensive operation, additionally there is only a small region of memory is set aside for quick memory accesses which is not large enough to store a large number of pre-calculated values, and the larger pool of global memory is much slower and not at all suitable for such a task. The shared or constant memory space is not big enough to store

LUTs for each atom type, but it is big enough to store the 12 parameters for all 103 elements used to calculate the potentials according to eq. (3.16). The potential is then calculated anew each time it is required for every atom and pixel but by utilising the *constant* memory of the GPU to store the parameters, we can ensure they are able to be accessed at high speed as the read will be broadcast to every processor in the multiprocessor which all require the same parameters at the same time as each multiprocessor loops through the exact same set of atoms. The full code listing with comments for the calculation of the transmission function can be found in Appendix A.1.2.

$$V_a(r,z) = 2\pi^2 a_0 e \sum_{i=1}^{3} \frac{a_{iz}}{r} \exp\left(-2\pi r\sqrt{b_{iz}}\right) + 2\pi^{\frac{5}{2}} a_0 e \sum_{i=1}^{3} c_{iz} d_{iz}^{-\frac{3}{2}} \exp\left(\frac{-\pi^2 r^2}{d_i}\right) \tag{3.16}$$

Parallelisation of the other components of the multislice algorithm was much more straightforward than the calculation of the transmission function, in the majority of cases the other code did not require any extensive optimisation due to the relative simplicity of the multislice algorithm. The benefits of the sheer performance of GPUs can provide large performance increases even in parts of the code that provide little room for optimisation. The numerous FFTs were handled by an external library which is already optimised for GPUs[2]. The rest of the code listings with commenting and short descriptions can also be found in Appendix A.

[2]cuFFT which is developed by Nvidia

An overview of the entire code is given in alg. 1.

Algorithm 1: Multislice Image Simulation

```
   /* use graphical user interface to select structure file and
      set imaging parameters                                    */
 1 if valid structure selected then
 2 |   upload list of atoms onto GPU;
 3 |   sort atoms into bins on GPU;
 4 |   rearrange atoms into order on host;
 5 |   upload sorted atoms to GPU;
 6 |   generate initial wavefunction on GPU;
 7 |   for slice = 1 to number of slices do
 8 |   |   calculate transmission function;
 9 |   |   multiply transmission function and wavefunction;
10 |   |   propagate new wavefunction over slice thickness;
11 |   end
12 |   /* at this point we have simulated the exit wave for the
      |      structure                                          */
13 |   create image by simulating the effect of lenses on exit wave;
14 end
```

3.4 Results

The results in this section demonstrate the ability of this multislice code to both reproduce the expected result when compared against other well established software, and also the ability to improve upon these existing results with the additional functionality available. Finally the improvements made by optimising the code for implementation on GPUs are demonstrated by benchmarking the code against other established software to measure the performance. The primary software used for the comparison is Kirkland's TEMSIM [Kirkland 2010] which is described in section 2.3.3. TEMSIM provides support for timing the execution of the simulation and has also been parallelised to improve performance on multi-core or multi-CPU systems, and so provides a more reasonable measure of the effect of GPU parallelisation than comparison with software with no form of parallelisation. The performance of the GPU based implementations of CMS, IMS, and finite difference multislice (FDMS) within this software package will also be compared against each other as well as against the external software.

3.4.1 Comparison of Simulation Methods

The changes made to the handling of atomic potentials in IMS, and also in FDMS, should in general improve the accuracy of multislice calculations, but these changes should be more apparent for some simulations than others. Simulations of perfect crystal structures that could evenly be divided into slices should not see much benefit, while amorphous structures and structures with defects should be much more accurately represented. These changes are also particularly important in the simulation of diffraction patterns, if the slice thickness cannot be chosen so as to match up with a lattice spacing in the beam-direction, the approximation of projected atomic potentials can lead to the generation of an artificial repeating potential with a repeat distance equal to the slice thickness which can create artificial higher-order laue zone (HOLZ) lines in the simulated diffraction pattern, which is undesirable. The removal of the minimum slice thickness limitation should also allow for much more accurate simulations to be performed should this be required. Furthermore, the addition of realistic noise should also for allow for improved simulations of objects which can only be imaged under relatively low dose conditions, and finally the addition of CCD characteristics should allow for simulations which can be more accurately compared to experimental images, and to simulate the benefits of imaging with one detector over another, e.g. comparison of CCDs to direct detection cameras or image acquisition with different levels of binning.

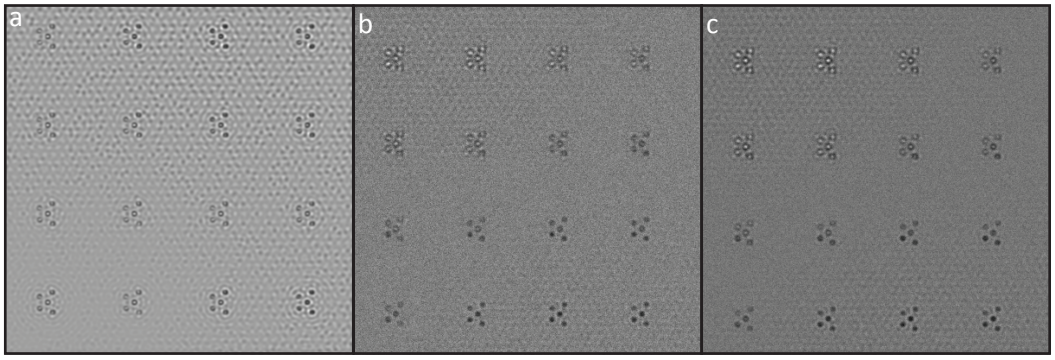

Figure 3.8: Comparison of 3 simulations of polyoxometalates supported on a corrugated graphene sheet using a) TEMSIM, b) Improved Multislice, c) Finite Difference Multislice. The height of the graphene sheet and decorated polyoxometalates is varied slowly across the field of view resulting in small height changes that are not accounted for in the described CMS method. The simulation was performed at around gaussian focus for an 80kV abberation corrected microscope with a spherical aberration of $1\mu m$

Figure 3.8 highlights some of the differences between the simulation methods described so far for a simulation of small molecules on a corrugated graphene

sheet. All three images are very similar, but the inclusion of electron dose leads to more realistic looking images for the later 2 images. The FDMS method should generate the most accurate exit wave as it fully accounts for the second derivative of the potential within the formalism, and is able to fully make use of very small slice thicknesses so the potential is very finely sampled in the z direction with no approximations made based on the 3D positioning of the atoms. The performance of the alternative methods can be evaluated by comparing the difference in the resulting simulated exit waves with respect to the finite difference generated exit wave which is assumed to be more accurate. Whilst the changes in the IMS method should provide a more accurate calculation, it is still difficult to produce simulations which match experimental images quantitatively with the source of the mismatch generally referred to as the Stobbs Factor [Hÿtch and Stobbs 1994]. Figure 3.9 shows that the improvements to the potential generation method improve the results of IMS when compared to CMS for the same slice thickness. For large slice thicknesses the number of samples used in the numerical integration of the potential is probably a limiting factor and could be increased to improve the accuracy at the cost of an even slower computation. Lowering the slice thickness of the IMS method improves its accuracy as demonstrated in fig. 3.10, however this approach is not applicable for the CMS method to due limitations over the suitable range of slice thicknesses which limit the final accuracy that can be achieved with this method. This effect can be seen in fig. 3.10 where the CMS method improves very little with lowering slice thicknesses in contrast with the IMS method which approaches the accuracy of the FDMS method as the slice thicknesses become comparable. As the slice thickness reaches a critical thickness the error rapidly increases for both methods, at larger slice thickness, each individual slice will contains atoms from more than a single layer within the structure which will significantly impact the quality of the results.

It can also be seen from the results in fig. 3.10 that the assumptions made in the transmission function calculation for CMS are generally quite accurate, at least when comparing the exit wave intensities produced using the two methods, the difference between the results of CMS and IMS in this case are very small unless the slice thickness is substantially lowered. Figure 3.10 also shows that the IMS methods produces results very much in line with those calculated using the FDMS method if a small enough slice thickness is used (still much larger than required for FDMS) and as such this is a good method for producing very accurate simulations without requiring quite as much computational time as the FDMS method (FDMS simulations typically vary between 1 and 2 orders of magnitude slower than IMS simulations).

It should also be noted that these tests are performed for a simulation of a perfectly aligned crystal structure (silicon [111]) which is also quite thick (50nm),

this is likely a best case scenario for the CMS method as the atoms are aligned into flat slices perpendicular to the electron beam direction naturally which is a condition that would otherwise be enforced by the CMS method anyway were it not the case. The relative alignment of slices of atoms should have no effect on the IMS method due to the changes to the potential calculations and so an aligned crystal structure was chosen so as not to bias the results towards either method.

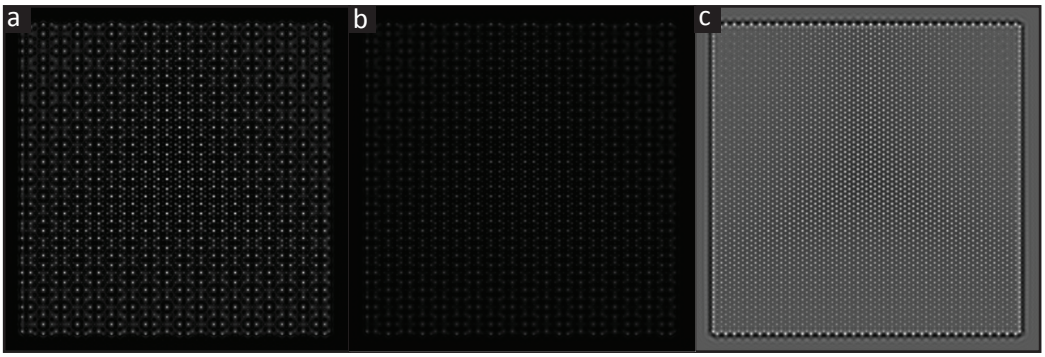

Figure 3.9: Difference maps comparing the accuracy of a) conventional multislice and the b) improved multislice method as described in chapter 3. Simulations though a 50nm section of silicon [111] were performed for a small slice thickness (0.15Å). Images a) and b) show the absolute difference between the FDMS exit wave and the corresponding CMS and IMS exit waves, the grayscale values are displayed on the same scale for both images with black being no difference indicating that the results using IMS where a much closer match to those using the FDMS method. Image c) shows the exit wave calculated using the FDMS method used to compare the exit waves for reference.

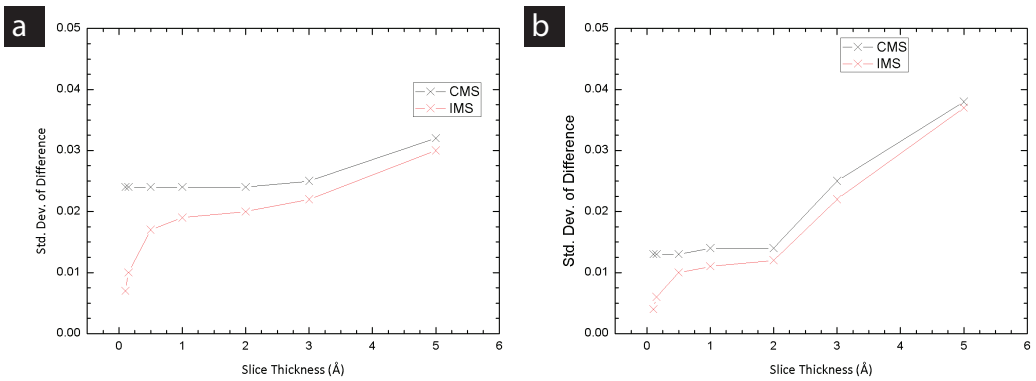

Figure 3.10: The standard deviation of the error per pixel between the FDMS generated exit wave and those from the IMS and CMS methods for different slice thicknesses for a simulation of a 50nm section of silicon [111]. a) shows the difference in the absolute value of the exit wave, and b) shows the difference in the phase of the exit wave. The error is reduced by taking smaller slices for the IMS method but this has little effect on the CMS method for thin slices. At values greater than a critical slice thickness the errors begin to increase quickly for both methods. The CMS assumptions make little difference for medium slice thicknesses but greatly reduce the computation time.

Figure 3.9 shows that the results calculated by the IMS method are closer to those from the more accurate FDMS method than the equivalent results using CMS when the simulations are performed for the same slice thickness. The trade off is that results calculated using the IMS method take slightly longer to calculate than using CMS, although still far less than using the FDMS method, (a detailed comparison of simulation speeds can be found in section 3.4.2. The IMS method is however quite flexible in that the slice thickness does not have to be chosen to match with the sample periodicity and so can be chosen to tune the computational speed. There are also other options available for tuning the performance of the IMS method, such as changing the number of samples used for integration of the atomic potential across the thickness of each slice, or changing the cutoff range at which potential contributions are included. The majority of time during the calculation of the transmission function is spent sampling the atomic potential from each atom numerous times and integrating the results and so changing the level of sampling has a large impact on the calculation time as can be seen in fig. 3.12 where the calculation times are plotted for the IMS method with both 10 and 20 potential samples per atom per slice, the end result is however very similar to simply halving the slice thickness which also doubles the sampling rate of the potential in the z direction, but also doubles the number of propagation steps and should therefore provide better results for a similar computational cost (the FFTs and propagation are inexpensive compared to the transmission function calculation).

After the exit wave has been simulated by the aforementioned methods, the next step is to simulate the effects of the lens system within the microscope which takes the exit wave and forms an aberrated image at a given defocus. The standard approach to this is simply to take the exit-wave in the frequency domain and multiply it by the lens aberration function which is parametrised by the aberrations of the microscope, and then to inverse Fourier transform this result to generate the image wave. This is a complex quantity which cannot be recorded using a CCD camera and so we take the modulus of this result to form the image. The addition of noise into the simulation process complicates matters slightly but is especially important to include when simulating images from objects which cannot tolerate exposure to high electron doses. Simulations containing no noise whatsoever can provide unrealistic estimates of image contrast where many of the image features in the simulation will be hidden well below the noise level in an experimental image. The actual amount of noise to be added depends critically on the electron dose as this will determine the number of electrons impinging on each pixel of the detector. Other noise components include the dark current associated with the CCD and also the readout noise associated with the CCD. The simulations shown in fig. 3.11 show the effect of varying the electron dose on simulated images of graphene and how this can be critical to understanding the images. In fig. 3.11 it is apparent the a relatively high electron dose would need to be used in order to clearly resolve the lattice at these imaging parameters.

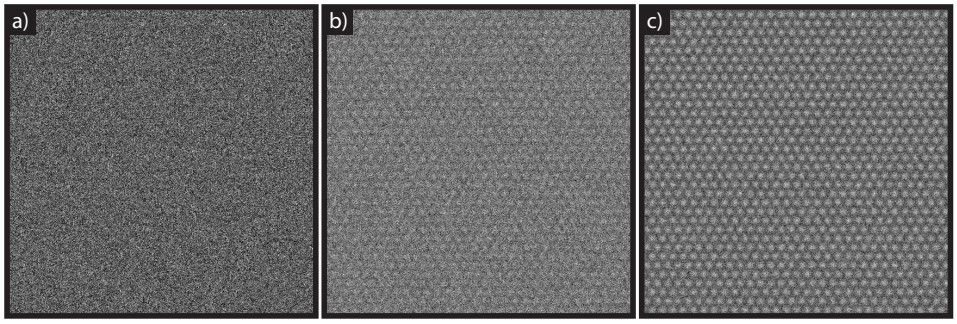

Figure 3.11: Multislice simulations of graphene with electron dose varying from a) $200e^{-}/\text{Å}^{2}$ b) $2000e^{-}/\text{Å}^{2}$ c) $20000e^{-}/\text{Å}^{2}$.

3.4.2 Benefits of Parallelisation

In order to test the effectiveness of rewriting the multislice code to be efficient on GPUs a series of comparisons were performed primarily against the Kirkland multislice code. The Kirkland code was chosen for comparison purposes as it is capable of reporting detailed data on the time taken performing various stages of the

calculation (the source code is also available for this project[3]), the Kirkland code has also been parallelised to run on multi-core CPU's and so provides a more meaningful benchmark for comparing the effectiveness of switching to GPU based computation than some of the other multislice software which only take advantage of single core processors. Efforts were undertaken to ensure that the simulations performed using the different programs were as similar as could realistically be achieved, this involved not assuming partial coherence as it is performed using different processes within the two simulations which would have a significant effect on the simulation time. Thermal displacements were also disabled in both pieces of software as both methods essentially involve resimulating the entire stucture a chosen number of times with small random atom displacements which should only introduce a linear dependency into the total simulation time. As well as comparing the speed to a CPU only software package, the speed of the various different methods, CMS, IMS and FDMS which have all been implemented on the GPU are compared against each other.

In fig. 3.12 we look at the change in performance as the size of the input structure is increased, in general for the CPU case the time taken for the simulation should scale approximately linearly with the structure size as for conventional multislice each atom only contributes once to the potential, and the number of slices also scales with the depth of the sample. In the GPU case it is not as straightforward, in many cases the code is limited by the latency in making memory accesses not actually performing arithmetic calculations, and so there is room to perform additional calculations whilst waiting for memory accesses to be retrieved which will not affect the overall time taken to complete a calculation. It is also entirely possible for smaller workloads that the GPU will not achieve a high utilisation and thus not be performing to its maximum capabilities, and so its performance may also be increasing as the problem size is scaled up. The efforts made in pre-sorting all the atoms into bins should also aid in the efficiency of the potential calculations as the structure size is scaled up, the pre sorted atom bins prevent loops of unnecessarily large numbers of atoms as each workgroup only loads the relevant subset of atoms. This performance scaling effect can also be seen when simulations are performed using only a small number of pixels, when the workload on the GPU is small it is not able to fully utilise its resources resulting in slightly worse performance for smaller problems than could otherwise be expected.

The performance of the code was benchmarked at different stages throughout the development cycle as various strategies were implemented to improve the calculation speed. The performance benefit of the individual refinements is shown in fig. 3.14 and shows that a large variance in performance that can be achieved through optimisation without affecting the actual simulated results.

[3]http://sourceforge.net/projects/computem/

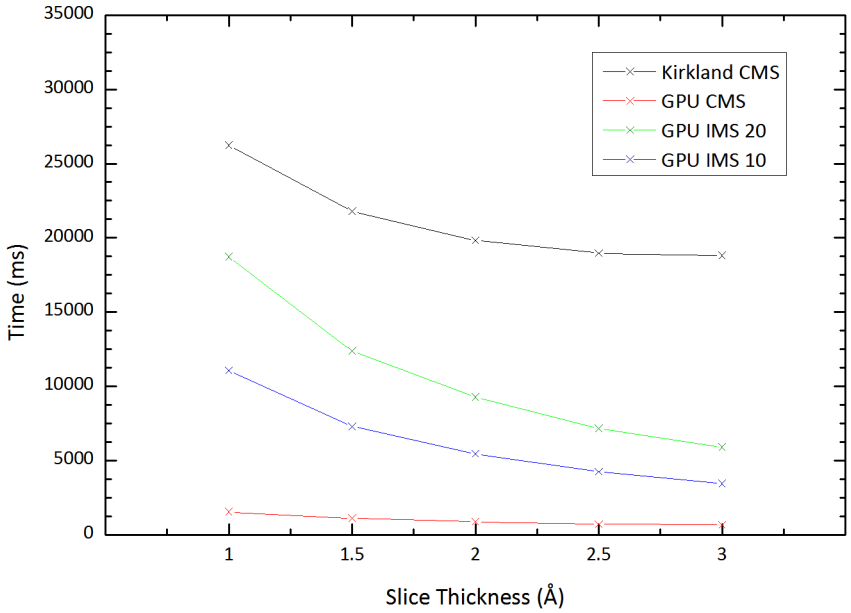

Figure 3.12: A comparison of the performance of various multislice methods such as Kirkland multislice and GPU based multislice methods for varying slice thicknesses. Images are simulated using a model of $\sim 200,000$ atoms of silicon. Here IMS10 and IMS20 refer to the IMS method using 10 and 20 steps respectively for the numerical integration of the potential in each slice.

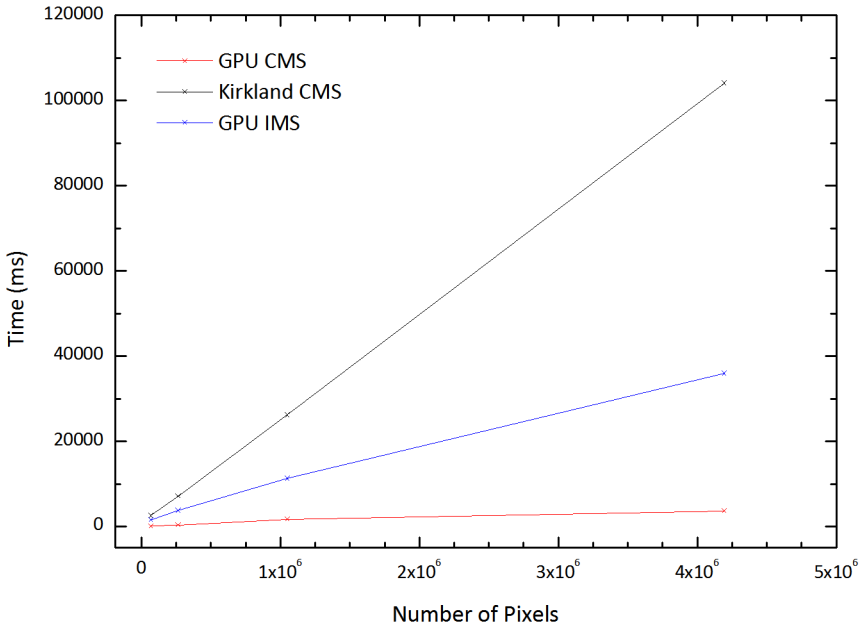

Figure 3.13: A comparison of the performance of various multislice methods such as Kirkland multislice and GPU based multislice methods for simulated images with varying numbers of pixels. Images are simulated using a model of $\sim 200,000$ atoms of silicon. Here IMS10 and IMS20 refer to the IMS method using 10 and 20 steps respectively for the numerical integration of the potential in each slice.

Figure 3.12 shows that GPU parallelisation in conjunction with the other modifications to the algorithm can drastically improve the time taken to perform multislice calculations using the CMS method. The GPU implementation of the CMS code completes 23x faster than the Kirkland CPU implementation averaged across all resolutions and slice thickness tested here. The minimum improvement between Kirkland's CMS and the GPU implementation was around $17x$ and the maximum around 38x. It is also evident from fig. 3.12 that the GPU implementation also benefits more from increasing the slice thickness than the Kirkland code, this is likely due to increased overheads from having to call more GPU kernels at low slice thickness which does not affect the Kirkland code and the fact that each atom features in a single slice in the Kirkland method no matter the chosen slice thickness. Figure 3.13 shows that the GPU implementation also outperforms the CPU only code by a much larger margin as the simulated image size is increased, this is because the GPU can schedule its resources more efficiently under larger workloads whereas the CPU performs relatively consistently across all workloads. At larger resolutions each multiprocessor on the GPU should also only have to include a smaller number

of atoms which may also have some effect on the overall efficiency.

The GPU based IMS code also outperforms the Kirkland multislice code in all of the tests performed here despite the IMS algorithm being more computationally demanding. In this case the use of GPU parallelisation can be seen to allow the use of a more sophisticated and accurate algorithm within the same time frame constraints as conventional software running the original multislice algorithm. The use of the finite difference multislice algorithm is still somewhat impractical however, the requirement to use thin slices to ensure stability means that slice thicknesses are usually on the order of 0.01-0.1Å therefore 1-2 orders of magnitude more slices are involved in the calculation. As no other suitable implementations of the finite difference method could be found to compare against, the finite difference method was also implemented in OpenCL so that the performance could be evaluated on both GPUs and CPUs as in table 3.1. It should be noted however that the finite difference method kernels have been designed to achieve high performance on GPUs and are not necessarily optimal on CPUs.

Device	Simulation 1	Simulation 2	Simulation 3
CPU	620129 ms	350690 ms	878874 ms
GPU	5697 ms	4094 ms	8057 ms

Table 3.1: Performance of the finite difference method multislice simulations tested on a multicore CPU and a GPU for various simulation sizes and resolutions. a) 14000 atoms at 1024×1024 b) 27000 atoms at 512×512 c) 20000 atoms at 1024×1024

Whilst the calculation times in table 3.1 are not entirely unreasonable for the GPU at under 10s for simulations of $\sim 20,000$ atom structures at 1024×1024 or below, it is far slower than the other options presented (Simulation 3 takes 27ms using GPU CMS), and the results are very similar to those obtained using the IMS method outlined here which does not have the same drawbacks. The results do however show an average speedup by a factor of just over 103× for the three simulations using the GPU. For more complex simulations such as STEM simulations or CBED patterns which require multiple runs through the same multislice algorithm, the extra time required for the FDMS method would push the calculation times beyond practical limits. The very high performance of the GPU based CMS and IMS routines however show promise for performing large scale STEM simulation on a desktop computer. A 512^2 pixel STEM simulation is essentially equivalent to performing 512^2 individual multislice calculations, in this case a single GPU CMS simulation for a $200,000$ atom structure at 512x512 pixel resolution takes 0.4 seconds, and so the full STEM image could be acquired in just over 29 hours. This kind of large scale STEM simulation would usually be done using a high performance computing cluster but here it is made feasible on a standard desktop computer by

taking advantage of GPU computing [Guerrero et al. 2008; Guerrero-Lebrero et al. 2010].

3.4.2.1 Kernel Optimisation

The performance of GPU kernels can vary quite substantially with very minor modifications to the actual code being executed, in some cases large performance changes can be found without even changing the kernel, just modifying the layout or shape of the different workgroups that are used to generate the results. In the specific case of the kernel used in the transmission function calculation, several modifications were made to the first working kernel that was written before the performance levels indicated above could be achieved. The steps that were undertaken to improve the performance of the working kernel without actually modifying the result are described below and the relative performance improvements with each optimisation are shown in fig. 3.14.

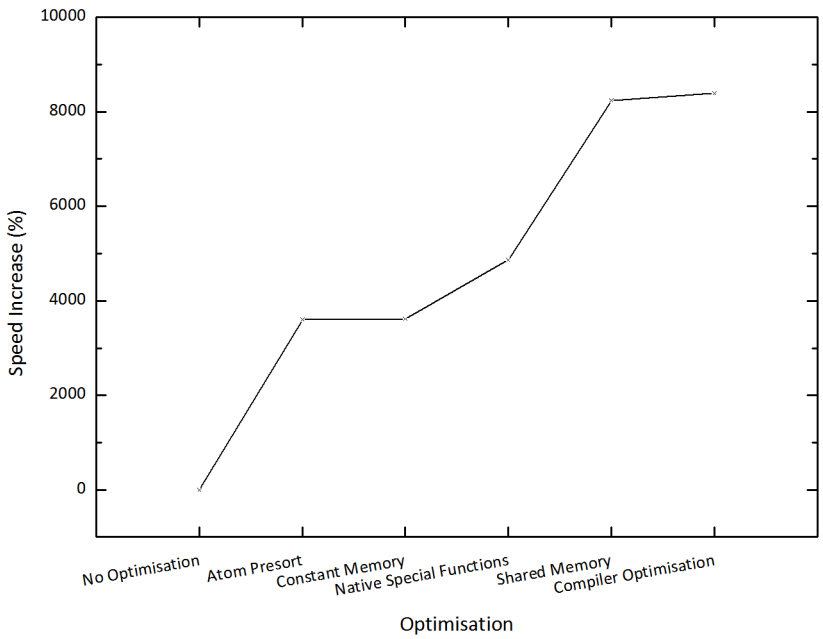

Figure 3.14: A comparison of the performance of the multislice simulation code as the transmission function calculation code is optimised. The initial design is badly suited to GPU computing and performs slower than the CPU version of the multislice code. The final version is over 80 times faster than the initial version. Images are simulated using a model of $\sim 200,000$ atoms of silicon.

The first version of the kernel did not feature any particular optimisation,

each pixel in every z-slice looked at all of the atoms that were located in that slice (this was determined via an initial sorting prior to this kernel being executed) and determined the potential for any atoms which were within the specified cutoff range. This kernel offered very poor performance as there were a very large number of slow memory accesses to global memory which were entirely unnecessary when loading atoms too far away from the current position to have significant potential contributions.

The first optimisation was to presort all of the atoms based on their spatial location within the slice, this was already done to organise the atoms by the z position but this was expanded to also assign groups to all the atoms in both the x and y directions. Each multiprocessor on the GPU would then perform a relatively simple calculation to determine which groups could possibly feature atoms which may have significant contributions to the potential. This offers a huge performance increase due to the considerably smaller memory bandwidth requirements as less data is requested from global memory when only relevant atoms are loaded. The number of groups that the atoms were separated into in both the x and y directions was also tuned to find the optimum performance. It is worth noting that the values offering optimum performance are likely partially related to the hardware being used and as such may not necessarily be the optimum values on different hardware, this is a limitation of the GPU computation model.

The next optimisation was to store many of the parameters in the *constant* memory of the GPU. The potential calculation for a single atom is characterised by a total of 12 numbers for each of the 103 atomic numbers for which they have been determined. These 12*103 numbers can all be stored in *constant* memory rather than *global* memory, this means that whenever multiple threads within a multiprocessor request a value from constant memory at the same time, the memory access only has to be performed once and can be distributed to all threads simultaneously. For data stored in global memory each thread must request its own separate copy of the variable, and these accesses will be performed in serial (unless the value is cached internally), this places a large demand on the available memory bandwidth which significantly impacts performance. It can be seen in fig. 3.14 that this particular change had almost negligible impact on the performance of the program, this suggests that these values were being cached automatically by CUDA and so this change was unnecessary.

GPUs usually contain hardware designed specifically for the computation of certain functions such as sin, cos, exponential, square root, log etc. This hardware implementation of the transcendental functions is not used by default however, they are usually calculated as they would be on a CPU using polynomial approximations which can be more precise but is usually much slower. Another large performance

increase was realised by switching to device native functions where possible as the calculation of the potential makes significant use of many of the transcendental functions.

A further optimisation was to restructure the kernel to more accurately map to the preferred memory access patterns of GPUs. The initial version of the kernel had each multiprocessor determine which atoms it needed to calculate the potential for, and then each thread within the multiprocessor would loop over the entire group of atoms, loading their positions from global memory separately to determine the radial distance and thus calculate the potential. This memory access pattern is suboptimal on a GPU because multiple threads in the workgroup are accessing the same memory element at the same time from *global* memory, this causes the memory accesses to be serialized which is therefore slow and inefficient. The preferred memory access pattern is for threads which are next to each other in the workgroup to access memory elements which are also next to each other when using global memory. This allows the memory accesses of adjacent threads to be grouped into one much larger memory access which is far more efficient.

To implement this memory access model, a *local* memory array was created for each workgroup, and once the workgroup had determined which atoms were required during the rest of the calculation, each thread would load a single atom into the *local* memory array. The atoms are stored in global memory in order of x-group then y-group then z-group via the presorting routine. This means all of the atoms in a whole row of blocks in the x-direction will all be stored contiguously in memory and hence can be loaded from *global* memory to *local* memory in single coalesced read operations. During the potential calculation, the atom coordinates are then retrieved as before, but from *local* memory instead of *global* memory, as *local* memory also supports broadcasted memory reads like the *constant* memory, there is no performance penalty for having multiple threads reading the same value from *local* memory at the same time.

The final optimisation is perhaps the most simple, once a kernel has been written, it first has to be compiled before it can be executed on the GPU. Modern compilers are quite capable of optimising code to make it run slightly faster but they are somewhat limited in scope to avoid introducing potential errors. There are some additional keywords that can be specified as part of the kernel which do not change the functioning of the code at all, but can be interpreted by the compiler to help it optimise the code more aggressively. By adding the keyword *const* which specifies that the values in an array will not change over the course of the kernel, and the keyword *restrict* which guarantees to the compiler that two different pointers will not point to the same object and the object will not be accessed via any other pointer, the compiler can make additional optimisations that can boost the performance of

the kernel.

Figure 3.14 shows the evolution of the kernel's performance as the various performance optimisations are added to the initial kernel. The final version of the kernel proved to be over 80 times faster than the initial kernel, despite them calculating the same result. The initial kernel however was very badly designed for use on a GPU due to the huge number of inefficient global memory reads making it slower than the CPU version at this point. The atom presorting was used to greatly reduce the number of global memory read operations before the later optimisation which mostly made the global memory read operations coalesced rather than serialized.

3.5 Chapter Summary

There is considerable room for improvement within the field of image simulation for TEM, the multislice algorithm has been shown to map nicely onto GPU hardware and several improvements enabled via GPU computing have been demonstrated which increase either the accuracy or speed of multislice simulation. These adaptations have all been implemented within a single image simulation software package pictured in fig. 3.15. The effects of changing image simulation variables on both simulation accuracy and speed has also been investigated for the various different algorithms. The choice of GPU based parallelisation also ensures that the software will scale in performance as newer and faster hardware is realised with more and more processing units without any modifications to the code being necessary. There is also still further scope to improve the performance of the current IMS implementation and a modified version of the simulation software has been released as open source for further development [4].

[4] https://github.com/ADyson/clTEM

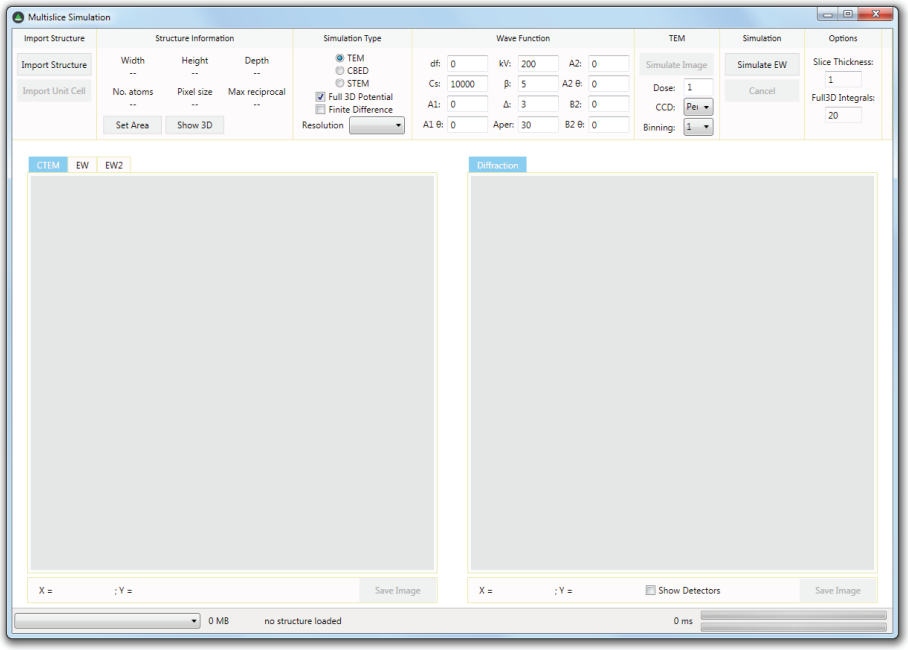

Figure 3.15: A screenshot of the multislice software GUI.

Chapter 4

Exit Wave Reconstruction Software Development

4.1 Introduction

The process of forming a phase contrast image within a TEM is a complex procedure, the varying specimen potential creates local variations within the phase of the exit wave, but in order to image these variations, they must first be converted to changes in amplitude so they can be detected by conventional means. This can be done in an analogous way to light microscopy using a phase plate which adds an additional phase shift to the scattered electrons, but ideal phase plates are still not available for electron microscopy [Zernike 1942; Danev and Nagayama 2001; Danev et al. 2009; Gamm et al. 2010]. Instead the phase shift induced by the aberrations of the imaging system are used to impart an additional spatial frequency dependent phase shift, this is described by the PCTF, eq. (4.1). This means is it is necessary to image with some degree of aberration in the system in order to generate any contrast for a weak phase object otherwise the PCTF would be zero. The downside of imaging with aberrations in the system is that they cause some delocalization of the contrast and can make the images hard to interpret directly because the aberration function causes contrast oscillations at higher spatial frequencies under the effects of defocus. There will also inevitably be spatial frequencies at which no information is transferred within each of the defocussed images.

EWR is a method that utilises a series of images, and using knowledge of the imaging conditions under which each image was taken, can be used to reconstruct a full complex wave where the information transfer is approximately uniform out to the information limit of the microscope rather than suffering from the oscillations of the PCTF. Information that was missing from one image can be recovered from a different image, and the noisy information at high spatial frequencies can be averaged

across all the images in the series to get a better estimate. EWR also recovers the full complex valued exit wave much like holography [Gabor 1948], this means the residual aberrations in the system can be numerically removed from the exit wave if their values can be accurately determined[Thust et al. 1996; Erni et al. 2010]. EWR is also a simple technique to implement practically as it does not require any additional hardware beyond computer control of the microscope focus which is standard on modern microscopes.

There are numerous algorithms presented in the literature for processing images to determine the exit plane wavefunction. An overview of the basic principles is given in section 2.3 along with a description of a few of the available algorithms. This chapter is mainly concerned with the specific implementation of the chosen method, FTSR, which is explained in detail in section 2.3.2. A plugin has been developed in C++ which operates within Digital Micrograph^{TM} (DM), a common software package developed by Gatan which is used to interface between microscopes and computers and provide computer control over the microscope operations, as well as image processing and scripting functionality.

4.2 Image Acquisition

A suitable series of images for EWR is one in which each image is taken under slightly different imaging conditions to ensure that across the entire series, all frequencies below the information limit of the microscope are represented. It is also important that the aberration conditions are accurately known, or can be determined, for each image in the series. In practice this is usually achieved by determining the majority of the aberrations using the aberration corrector alignment procedure of an aberration corrected microscope to characterise the imaging conditions immediately prior to acquisition of an image series. The limited stability of the aberrations for TEM imaging correctors means the measured aberrations coefficients will only be representative of the actual aberration values for a small time frame after they are measured meaning this process should be repeated for subsequent focal series as lower order aberrations are most susceptible to fluctuations [Barthel and Thust 2010].After the aberration conditions have been determined a series of images is then acquired whilst adjusting only one of the aberrations, usually the defocus. On microscopes without an aberration corrector the same techniques can still be applied manually to determine the aberrations (see section 2.5), or a reconstruction can be performed without accounting for some of the aberrations resulting in an exit wave with some residual aberrations still present.

The information content of a single image taken under phase contrast conditions is given by the PCTF, this is an oscillatory function, and where it is zero-valued,

no information is transferred to the image at that spatial frequency. The simplified PCTF assuming only defocus Δf, and spherical aberration C_s is given in eq. (4.1), it is assumed that the astigmatism should be close to zero for acquisition of a focal series. By recording images at different values of defocus, the position of the zero crossings of the PCTF are modified so that information content missing from one image may be contained within another image at a different defocus as shown in fig. 4.1. The same effect would be possible by adjusting other aberrations, such as the C_s or two-fold astigmatism $A1$ etc. however for practical purposes it is easiest to adjust the defocus as this can be achieved with fine control in most microscopes, and the defocus can also be adjusted over a very large range of values without significant impact on the other aberrations. The information transfer is also affected by spatial and temporal coherence, these have the effect of damping the information transfer at higher frequencies so that beyond a certain spatial frequency known as the information limit, the information transfer is below the noise level in the images.

$$PCTF(\mathbf{k}) = -\sin\left(\pi\lambda\Delta f^2 + \frac{\pi C_s \lambda^3 \mathbf{k}^4}{2}\right) \tag{4.1}$$

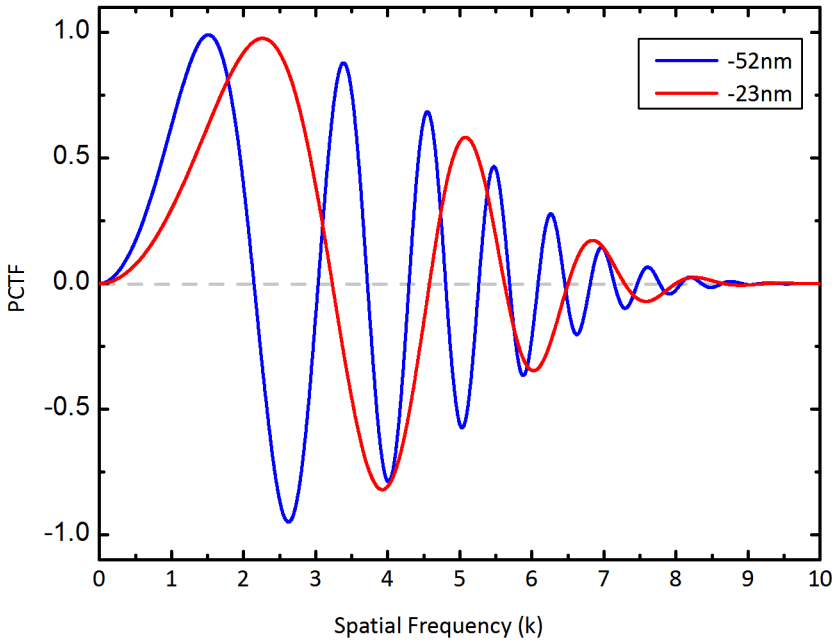

Figure 4.1: The PCTF for $\Delta f = -23$ and -52nm and $C_s = 1$um. The effects of spatial and temporal coherence have also been included as dampening envelopes which have the effect of reducing the single to zero past a certain frequency, this is the information limit of the microscope.

The image acquisition routine developed as part of this plugin can be used to automatically acquire a suitable focal series for reconstruction. Manual input from the user is required to move the specimen to an area of interest and then adjust the focus and astigmatism close to zero before acquisition of the series. The focus increment between images can be specified via the user interface and the plugin will determine the appropriate number of DAC units to adjust the focus by (provided that the calibration between defocus and DAC units has been determined). The number of images to take and additional settings such as binning and exposure time can be entered via a GUI which is shown in fig. 4.15. Upon starting the acquisition, the microscope will automatically adjust the focus to the position required for the first image, and then procedurally acquire the rest of the series at the specified defocus step before returning to the original focal position. Using computer control of the focus allows this to be done much faster than could be achieved by manual adjustment of the focus and with greater accuracy, the speed of the acquisition is important as the actual focus of the lens system and the 2-fold astigmatism will also drift away from the specified value over time [Barthel and Thust 2010]. A dwell

time can also be introduced if necessary to allow for a pause between each image acquisition, the electron beam can also be effectively blanked during the downtime between acquisition of consecutive images by using beam tilt to direct the beam over a different area of the sample, this can help to lower the total exposure for beam sensitive samples. All information about the acquisition conditions are then stored within the images themselves so that when they are selected for reconstruction the conditions do not have to be re-entered, even on a different computer as long as it has the plugin installed.

4.2.1 Voltage Induced Focus Variation

In addition to the standard image acquisition method outlined in section 4.2, in microscopes with the necessary hardware installed, the defocus of the image can also be adjusted by fine control of the electron acceleration voltage V. A small change to the accelerating voltage can be achieved in most microscopes that are fitted with electron energy loss spectroscopy (EELS) detectors, since this adjustment is often used for alignment of the spectrometer. Accurate computer control over this small accelerating voltage offset can be used to enact a focus change without adjusting the strength of the objective lens which is primarily used for focussing the microscope. The electron wavelength is modified when the accelerating voltage is adjusted as shown by eq. (4.3) where m_0 is the electron rest mass, e is the electron charge and c is the speed of light. This change in wavelength causes the electrons to be brought to a different focus due to the chromatic aberration in the objective lens, chromatic aberration describes the inability to bring all wavelengths to the same focus. Chromatic aberration correction in TEM is not widely available and so its presence is exploited here to enact focus changes. The focus change Δf for a change in voltage ΔV and a chromatic aberration C_c is given in eq. (4.2) [Hosokawa et al. 2013].

$$\Delta f = C_c \frac{\Delta V}{V} \left(\frac{\frac{eV}{m_0 c^2} + 1}{\frac{eV}{2m_0 c^2} + 1} \right) \tag{4.2}$$

$$\lambda = \frac{1}{\sqrt{1 + \frac{eV}{2m_0 c^2}}} \tag{4.3}$$

This mode of operation can be beneficial for a number of different reasons, the main benefit being that the voltage change can be done in very small increments, this should allow for very small focus steps, smaller than the minimum increment achievable using the traditional focus change method, this method also allows us to use custom focal steps which are not multiples of the focal step increment of the microscope objective lens. It may also be beneficial for the stability of the lens aberrations if the lens current does not often need to be changed as it is not required

for focussing and remains unchanged throughout the entire focal series acquisition.

4.2.2 Free-run Camera Mode

The image acquisition script was also developed to enable the different image acquisition modes on the CCD where possible, in the standard mode, the CCD is triggered to perform an acquisition at a specific moment and after each image exposure the CCD is unable to record another exposure whilst readout of the previous image is being performed, in this method a substantial delay is experienced between the capture of successive images. This extra time incurred is problematic when looking to capture of series of images of something that is time-sensitive as the total time taken to record the series is far greater than the sum of the individual exposures depending on the CCD readout rate. The beam blanking protocol mentioned in section 4.2 prevents problems from excess exposure to the electron beam but we can substantially improve the rate of focal series acquisition by operating the CCD in such a way that it is continuously acquiring images and transferring the data back to the computer without pause. This is much faster than acquiring all the images separately, but you lose the ability to trigger the acquisition at a given time as a trade off. This means some acquisitions will be during the time in which the microscope was making adjustments to the focus etc. and have to be discarded, and we also cannot blank the beam any more as the acquisitions are happening continuously. Having access to both modes of operation makes it possible to select the most appropriate mode for the requirements of the individual sample.

4.3 Image Registration

After the acquisition of an image series suitable for EWR, the first step of the reconstruction process is to align the images in the series with respect to each other. This is a complex problem as although the images are all of the same subject, in most cases the images will have drifted laterally between exposures and the focus has been changed between each image which can cause significant changes in image contrast which can cause problems for some alignment procedures. In the case of small focus steps the consecutive images retain a large degree of similarity with each other and so traditional image registration procedures like cross correlation can be quite effective to find the lateral image displacement. If the focal step between images is large however, the images may not have much similarity at all and so more sophisticated techniques to align the images are required that can either account for the change in focus or are unaffected by it. If the specimen drift between images is too large then it is likely that the drift during the image exposures may be detrimental to the reconstruction even if drift between images is accounted for. There are also

numerous other sources of misalignment that may need to be accounted for, there may also be a change in magnification between images that is also dependent on the size of the focus change, for small changes of focus on the order of 20nm or less this magnification change proved to be negligible but for larger changes of focus the effect became increasingly prominent. There can also be a focus-dependent image rotation, although again this is only really prevalent when using large focus changes. The image alignment procedure is of critical importance to the reconstruction process and so the images must be aligned to a very high accuracy (sub-pixel) in order to generate a good reconstruction. It is also important for the general applicability of the reconstruction software to be able to deal with the magnification and rotation to ensure the EWR routine can be used in as many cases as possible.

4.3.1 Phase Correlation and Mutual Information

The software developed here uses two main methods to correct the translational motion between images within a series, the first is the phase compensated phase correlation function (PCPCF)[Meyer et al. 2002] and the second is a technique based on MI[Viola and Wells III 1997], the two separate image registration methods can be easily switched between as one may prove more useful than the other for certain series of images as will be discussed later. Both of the techniques have been implemented utilising GPU parallelisation in order to make the reconstruction process as fast as possible towards the ultimate goal of producing exit wave reconstructions in real-time, this is particularly important in the case of MI which can be time consuming otherwise.

4.3.1.1 The Phase Compensated Phase Correlation Function

The PCPCF method is related to perhaps the most commonly used method of image registration, the cross correlation function (XCF) [Brown 1992]. The XCF returns a measure of the similarity between two images as one of the images is displaced with respect to the other. This method can be computed in the frequency domain using FFTs which makes it much faster than calculating a convolution in real space. A potential downside of registration methods that are computed in the frequency domain is that the FFT assumes periodic boundary conditions which leads to wrap-around effects from the edge of the images, these effects can be minimised somewhat by the inclusion of windowing functions or image padding but it can be difficult to remove them completely, however calculating correlations via convolution in real space is very time prohibitive. It is also difficult to perform a normalized cross correlation in the frequency domain [Lewis 1995] so this is usually done in the

spatial domain which is much slower. The XCF is defined as follows:

$$XCF = FT^{-1}\left(FT\left[I_1\right]^* FT\left[I_2\right]\right) \tag{4.4}$$

Where FT denotes a Fourier transform. We can also define the phase correlation function (PCF) which is related to the XCF:

$$PCF = FT^{-1}\left(\frac{FT\left[I_1\right]^* FT\left[I_2\right]}{\left|FT\left[I_1\right]^* FT\left[I_2\right]\right|}\right) \tag{4.5}$$

where I_1 and I_2 are the intensities of image 1 and image 2 to be registered with respect to each other. The result of performing these correlations is an image with the same dimensions as the original input images, where the value at each pixel is representative of the degree of correlation between the two images for a particular integer pixel image displacement. Usually the correlation will be low valued everywhere except at one place which is the correct image displacement to the nearest pixel. The PCF is commonly preferred to the XCF for image registration in electron microscopy as the XCF can be dominated by contributions from frequencies which are dominant in the Fourier transform (e.g. in images containing crystalline structures where the XCF may find a series of peaks due to the translational symmetry of the crystal), this effect is reduced in the PCF by setting the modulus of the transform equal to unity for all frequencies [Meyer et al. 2002]. The PCF and XCF are both quite robust in the presence of noise which is a general feature of frequency domain methods [Sarvaiya et al. 2012]. As the techniques are both performed in the frequency domain, there can be considerable artefacts due to the wrapping of the image at the boundaries which become more significant for larger image displacements, though this can be lessened by windowing the images in real space beforehand. The XCF and PCF can both be computed very quickly even for very large images as they require little more than 2 forward FFTs and one inverse FFT followed by a search over the resulting image for the peak/maxima, all of which can be easily parallelised.

Image registration for the alignment of focal series is somewhat different to the traditional case of image alignment where we are trying to align two images of the same subject with perhaps just an translation between the two images. In focal series alignment we are trying to align two images of the same subject, but the imaging conditions have also been adjusted. For small changes to imaging conditions the images will still maintain a high degree of similarity, but if the change is more significant, e.g. a very large focus shift, the XCF and PCF may fail entirely to align the images. Another technique heavily related to the XCF and PCF is the PCPCF, here another term is introduced to the correlation to directly account for the changes in image contrast between the images that should result from a specified change in

defocus. In the case of a change in defocus only Meyer et al. showed that the phase change between the two images can be compensated when only the image defocus difference, D, is known which produces a sharp correlation peak at the correct image displacement [Meyer et al. 2002]. When the defocus difference accounted for is not correct, the correlation peak will appear as a series of concentric circles centred around the correct image displacement. Examples of image alignment of the same pair of images using the 3 techniques listed is shown in fig. 4.2. The PCPCF for the correct defocus collapses to a very sharp peak whereas the other methods both produce a broader peak, in the case of the standard PCF the highest correlations are located in a circle around the correct image displacement and so the image would be misaligned by simply finding the position of the maximal correlation value.

$$PCPCF = FT^{-1}\left(\frac{\cos\gamma_D \cdot FT\left[I_1\right]^* FT\left[I_2\right]}{\left|\cos\gamma_D \cdot FT\left[I_1\right]^* FT\left[I_2\right] + h\right|}\right) \qquad (4.6)$$
$$\text{with } \gamma_D = \pi D\lambda\left|\mathbf{k}\right|^2$$

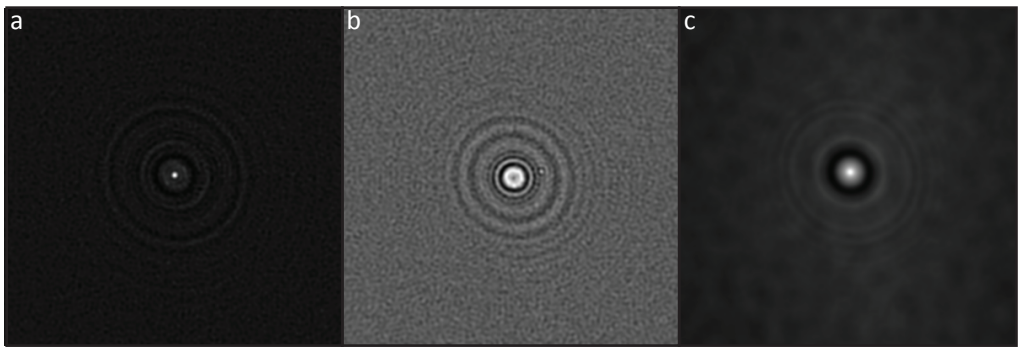

Figure 4.2: Results of performing a) PCPCF b) PCF and c) XCF on two images of the same specimen taken consecutively with a intentional defocus change of approximately 80nm. These images map the strength of the correlation between the two images as a function of image displacement with the position of the peak indicating the best alignment between the images.

The fact that the correlation peak is only sharp when the correct defocus difference is used allows for the height of the correlation peak for different defocus differences to be used as a means to determine the correct defocus difference between two images. By calculating the PCPCF for a range of different defocus differences, the peak height will reach a maximum for the correct defocus difference giving us both the image displacement and defocus difference for each image pair [Meyer et al. 2002].

4.3.1.2 Mutual Information Based Image Registration

Image registration using MI is an entirely different method of image alignment from the cross correlation based techniques in section 4.3.1.1. MI is an entropy based method which proves to be very robust with respect to variations in illumination, noise, contrast and has many other features which make it ideal for the alignment of electron microscopy images. MI has long been used in medical imaging registration as a means to solve the 'sensor-fusion' problem, that is to register images that have been recorded on different sensors or different pieces of equipment or at different magnifications [Shams et al. 2010a; Cahill 2010; Viola and Wells III 1997]. At heart, MI is a measure of how much is known about one image, given a different image, MI is defined in eq. (4.7).

$$MI(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left[\frac{p(x,y)}{p(x)p(y)} \right] \tag{4.7}$$

Where X, Y are the two images, $p(x), p(y)$ are the probabilities of a pixel having value x,y in each image respecitvely, and $p(x,y)$ is the probability of a pixel having value x in X and value y in Y. This can equally be written in terms of the entropy as in eq. (4.8).

$$MI(X;Y) = H(X) + H(Y) - H(X,Y) \tag{4.8}$$

Where the entropy is defined as in eq. (4.9)[Shannon 1948], where $H(X)$ and $H(Y)$ are the marginal entropies of the individual images and $H(X,Y)$ is the joint entropy of the two images.

$$H = -\sum_{i=1}^{n} p_i \log p_i \tag{4.9}$$

The probability of a particular value occurring, p_i, for each image can be calculated through the use of a histogram, and for the joint entropy $H(X,Y)$ a joint histogram can be computed to determine $p(x,y)$. In essence, the mutual information will be highest, when all pixels that have a value I_1 in one image, have a value I_2 in the second image, irrespective of any particular relationship between the values of I_1 and I_2 in each image, in this way, even if we had a complete contrast reversal in the electron microscope, and all objects that were previously light (high-valued) become dark (low-valued), the MI would be entirely unchanged as long as all the pixels of the same values were transformed in the same way. By using a limited number of histogram bins to determine the probability distributions this allows for the small variations in each pixel value caused by noise whilst still providing enough differentiation between the pixel values to allow for image registration. The number of image bins for the histogram can be varied to suit the individual image series if required.

The 2D joint histogram is formed by having all the histogram bins from one image along the x-axis, and all the histogram bins from the other image along the y-axis, then for a pixel (i, j) from both images, the histogram bin from image 1, $h1$, is calculated and the histogram bin for image 2, $h2$ is calculated, then the joint histogram bin $(h1, h2)$ is incremented. Several joint histograms between an image and itself with an intentional image displacement introduced are shown in fig. 4.3, it is clear that when the two images are identical, the joint histogram consists of a narrow line as all pixels of a given value in image 1 all share a value (the same value) in image 2, this distribution has a low entropy. When the image displacement is small, the values from the first image should still be very close to the values from the second image and so the histogram still has fairly low entropy. As the image displacement becomes larger the image intensities become uncorrelated leading to a high entropy configuration.

Any transformation that affects equal valued pixels identically will only cause a respective translation in the joint histogram without affecting its entropy and thus leave the MI unaffected, this is one of the major benefits of this method. Because no specific relationship was specified between the actual values I_1 and I_2 in either of the two images, the technique is perfectly capable of aligning images taken using completely different equipment, i.e. aligning a scanning electron microscope (SEM) image to a TEM image or atomic force microscopy (AFM) image. It is also insensitive to some changes over the entire image, such as the level of illumination which can also be beneficial.

Just calculating the MI for a pair of images at a single displacement however does not tell us anything about the potential image alignment, the MI must be calculated for all possible alignments that are to be tested, and then the values of the MI for a particular displacement are compared against each other to find the displacement with the highest MI. Without an initial guess for the image alignment, this can require a large number of MI calculations which can be time consuming.

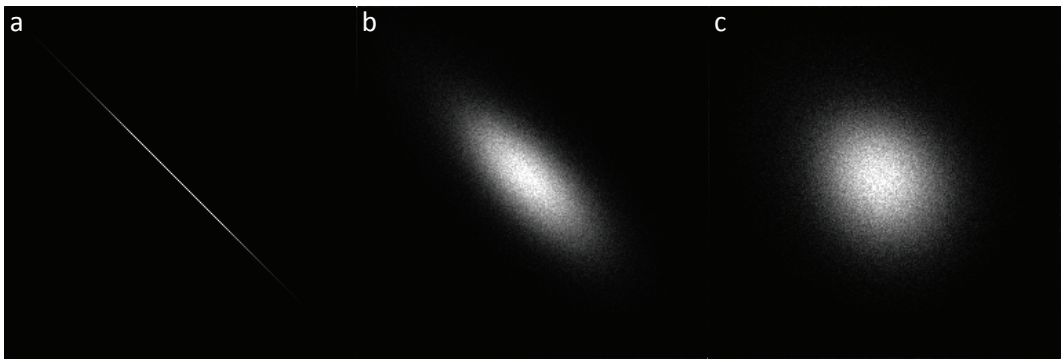

Figure 4.3: Joint histogram between an image and itself for intentionally applied image displacements of a) 0px, b) 1px and c) 5px. The joint histogram spread gives a measure of the correlation between intensity values in each image. The histogram was performed with 256 bins for each image totalling 65536 bins.

In the specific case of focal series image alignment using MI, the two images to be registered are not just identical images with an unknown image displacement, the change in imaging conditions between the acquisitions will lead to a change in image contrast between the images, however if these changes in image contrast retain the same relationship between like-valued pixels in each of the images, (i.e. pixels that have the same value as each other in one image, also have the same value as each other when the defocus changes), then MI will still provide a useful measure of the image alignment even in the presence of varying aberrations. Figure 4.4 shows the joint histogram between two images from a focal series for varying image displacements, the difference between joint histograms is pronounced for a displacement as large as 5 pixels but cannot be determined by eye for just a single pixel displacement, however calculating the entropy for the joint histograms will differentiate between the two cases making it suitable for image alignment of focal series.

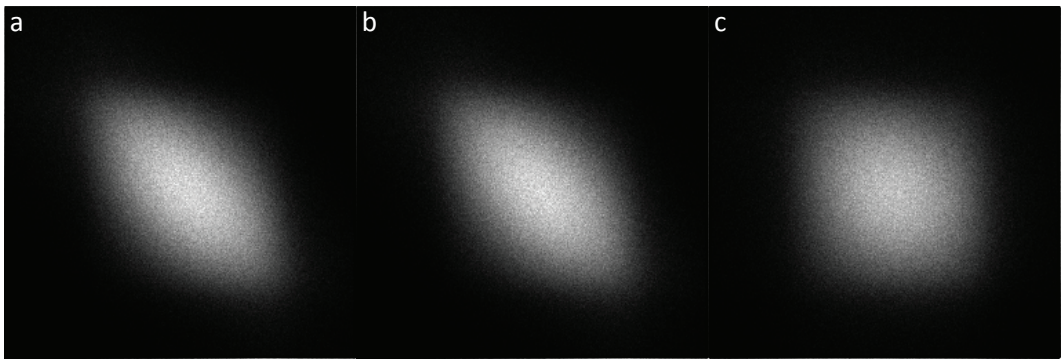

Figure 4.4: Joint histogram between two consecutive images within a focal series for an image alignment mismatch of a) 0px, b) 1px and c) 5px. The joint histogram spread gives a measure of the correlation between intensity values in each image. The histogram was performed with 256 bins for each image totalling 65536 bins.

The downside of this technique in relation to the PCPCF technique is the huge difference in computational effort required to calculate the MI over any of the XCF based methods. Whereas the correlation methods require little more than 3 Fourier transforms and then a search to find the peak location, the mutual information requires computation of a histogram for each of the individual images, and then a separate joint histogram for every possible image displacement to be tested. The MI method cannot currently be used to determine the defocus difference either. The single image histograms can be calculated by integrating across the joint histogram horizontally and vertically to save computing them separately, but this makes very little difference overall as the joint histogram calculation takes the majority of the calculation time and the single image histograms only need to be calculated once anyway.

In order to exhaustively test all the image displacement possibilities using MI on par with the FFT based methods would require an enormous computational effort (a MI calculation for each pixel) which is hard to justify, for this reason it is assumed in this plugin when using MI that the image displacement will be smaller than some defined value D_{max} and the MI is only calculated for displacements less than this value. This is a limitation when the drift value is assumed to be large as the computational time will scale with D_{max}^2. A more intelligent scheme could also be designed in which a coarse search is performed first over a larger area (i.e. one MI calculation for every 2 or three pixels) followed by an accurate search in the region of the first maxima, but this approach was not taken here. An alternative method would be to approximate the displacement using one of the XCF based methods and then refine this result using MI however the MI is often used for series where the XCF based methods fail.

4.3.2 Improved Sub-Pixel Accuracy

EWR usually necessitates aligning the images to an accuracy of better than one pixel in order to produce a good reconstruction. There are several ways that this accuracy can be achieved using the previously described techniques for image alignment. For the XCF based methods, finding the highest valued pixel will obviously only return an image displacement to the nearest pixel which may not be good enough. If the image Fourier transforms are zero-padded in reciprocal space up to twice their original size before being inverse Fourier transformed, then this is effectively the same as interpolating the images in real space to create an image with twice as many pixels in each direction with interpolated values inserted between the original pixel values. If these zero padded FFTs are then used as the basis for an XCF or PCF then when we find the pixel of maximum value in the XCF or PCF which corresponds to the maximum correlation, the doubling of pixels allows to to find the result to an precision of half a pixel with reference to the original images. This approach is however not very scalable as the double size FFTs require more memory to store the data and take much longer to compute which puts a practical limit on the size to which the result can be upsampled.

An alternative method to achieve sub-pixel precision is to perform a small peak fit to the correlation values surrounding the maximum location, by fitting the data in the x and y directions to some peak function, we can use the maximum position of the fitted peak as the location of maximum correlation. This can be further simplified to just using the points directly adjacent to the maximum in the x and y directions to form a 3-point parabola in each direction. The vertex of the parabola can be calculated straightforwardly from these values and will give a value to sub-pixel precision based on the strength of the correlation in different directions away from the maximum. This can also be applied in conjunction with the previous method in order to refine a result that has already been calculated to sub-pixel precision via interpolation.

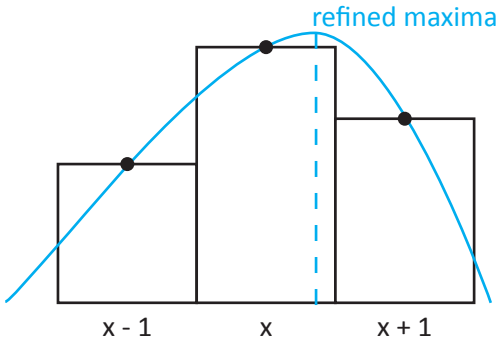

Figure 4.5: Refining the peak position from the maxima and adjacent data points by calculating the vertex of a parabola formed from the three data points.

The zero padding interpolation method above can be extended by zero padding the frequency domain transforms of the images to much more than double there original sizes, increasing the size of the images by an up sampling factor κ will improve the precision of the result by $1/\kappa$, however it is impractical to calculate Fourier transforms of such large images if a very high precision is required due to both time and memory constraints. For example, to register a typical electron microscope image of around 2048*2048 pixels to an accuracy of $1/10th$ of a pixel requires computation of at least one 2D Fourier transform of a 20480*20480 image, this is not really practicable without dedicated hardware, storage alone of one image of this size would require approximately 3GB of memory. It is however possible to compute the Fourier transforms with very high upsampling factors by calculating the discrete fourier transform (DFTr) using a matrix multiplication method[Guizar-Sicairos et al. 2008], whilst the FFT is a very fast and efficient way to calculate the cross correlation, it is only possible to calculate it for all possible displacements simultaneously which is troublesome for large transform sizes. The matrix multiplication method allows for calculating the cross correlation results for only a very small range of displacements instead of calculating the entire array of values, but using a less efficient method, this method remains practical even for large κ with much lower memory usage and calculation time if the region to be calculated is small enough. If we use the standard method to calculate the result to the nearest pixel, and then use the matrix multiplication method to calculate only the results for a very small region around the original estimate, we can achieve very high accuracy without needed to store huge amounts of data or calculate any large Fourier transforms. For an up-sampling ratio κ and original image dimensions of N by M with $N \leqslant M$, the algorithm is reduced to complexity $\mathcal{O}(NM\kappa)$ from $\mathcal{O}\left(NM\kappa\left[\log_2(\kappa M) + \kappa \log_2(\kappa N)\right]\right)$. The memory constraints are also greatly reduced as there is no need to form a image with dimensions κM by κN [Guizar-Sicairos et al. 2008].

For the MI method, it is possible to calculate the MI for any arbitrary image displacement if the images are sampled using some form of interpolation during the generation of the joint histogram. This allows the MI to be calculated to arbitrary levels of precision. In this software joint histograms for non-integer pixel displacements are calculated by taking the value from an image at a non-pixel position using the bilinear interpolation scheme as depicted in fig. 4.6. Although it is possible to calculate the MI for any possible arbitrary image displacements, the time taken to calculate the MI for each image displacement makes this somewhat prohibitive without using some coarse-to-fine search scheme as described earlier. For this reason, in this plugin the MI is only calculated for integer pixel displacements, and then the resulting peak position in the MI map is refined by the three point parabola method in fig. 4.5 to improve the accuracy.

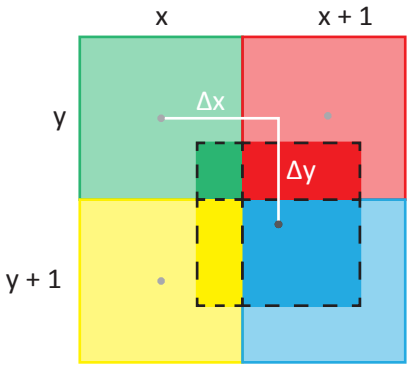

Figure 4.6: Sampling an image at fractional pixel position $(x + \Delta x, y + \Delta y)$ using bilinear interpolation. In bilinear interpolation the value for positions that do not coincide with an actual pixel are formed by taking the value of the 4 pixels they overlay weighted by the area of the overlap.

4.3.3 Compensation of Magnification and Rotation Changes

It is possible to automatically determine the changes in rotation and magnification using similar methods to those used for registering translational motion between images[Reddy and Chatterji 1996; Wolberg and Zokai 2000]. The automatic method consists of performing a log-polar transform of the original images $(x, y) \rightarrow (\log r, \theta)$, this log-polar transformed image can then be registered to another log-polar transformed image, however in this coordinate system, a change in angle is mapped onto a translation in y, and a change in scale is mapped onto a translation in x, $\log(ar) = \log(a) + \log(r)$, the image displacement must then be found using another registration step on a rescaled and rotated version of the original images. In tests with TEM images this process did not yield reliable enough results for the rotation

and scale leading to improperly aligned series.

For the purposes of this plugin, support for changes in scale and rotation is provided through the optional input of rotation and magnification calibration parameters, these are not determined automatically as part of the image alignment routine as the automatic procedure was not reliable enough. The changes in both rotation and magnification due to changes in the defocus are at least to an approximation linear, and so by entering the change in magnification and rotation per nm, we can determine the change in magnification and rotation between each image in the series from the 2 calibration parameters and the focus step between the images. If we assume one of the images to be at the correct scale, we can generate a new series of images where each other image is formed by interpolating the original rotated and scaled image and simultaneously rotating the image about its centre point. This leaves us with a series of images that are all at the same magnification and with no rotation between images. This partially corrected image series can then be used in the original focal series alignment algorithm to correct for the remaining translational motion between the images separately.

The calibration for change in scale and rotation is hard to measure accurately using manual methods without some crystalline reflections in the diffractogram to use as a guide. Figure 4.7 shows diffractograms from two images in a focal series of graphene oxide taken over a total defocus range of $400nm$. This focal series exhibits a change in magnification as highlighted by the positions of the crystalline diffraction spots in the two images shown. The crystalline diffraction spots enable measurement of the angle and spatial frequency of a particular reflection in 2 separate diffractograms and by dividing the change in angle/distance by the change in focus between the images the calibration can be determined. If there is no suitable feature in the frequency domain, then the plugin has an additional functionality to automatically determine the scale difference between a pair of images at different defocus. This is done by creating a copy of one of the images at various different scales via bilinear interpolation, and then measuring the correlation peak height against another image in the series via PCPCF for each of trial scales. The image correlation will be higher when the image scales are matched leading to the correct scale calibration parameter. This can then be input into the reconstruction software for the full focal series as this scale calibration is not performed as part of the reconstruction procedure and must be done separately. The same process could also be used to determine changes in rotation, but the rotation change has so far been negligible in comparison to the scale change, and to include both factors at the same time would increase the computational time significantly.

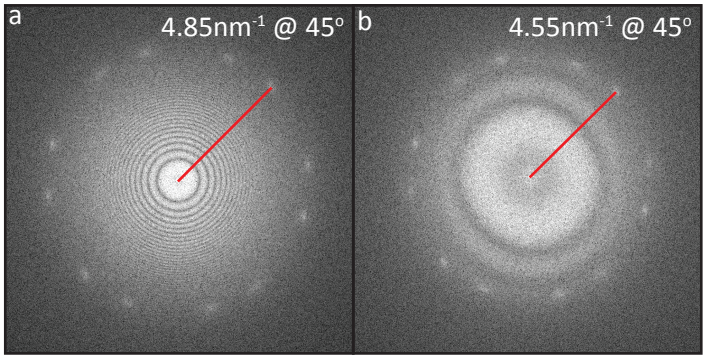

Figure 4.7: Measuring the position of crystalline reflections in diffractograms for two images of the same sample taken at a different defocus can in some cases show a considerable change in magnification, in this case the change is 6.5% over the full range of a 20 image focal series with 20nm focus steps. The angle of the reflections is constant indicating there is negligible change in rotation to account for within this series. The change in magnification is likely a result of having a non-parallel electron beam.

4.3.4 Iterative Alignment

The typical procedure to align a series of images, is to pick a single image, or a sub-region within an image as a reference area, and then align the other images with respect to this reference area. This can be done by directly registering each image to the reference image, or instead by registering each image to an adjacent image, and then calculating the misalignment with respect to the reference image by accumulating the displacements of all the images between this image and the reference image. In many cases it can be problematic to register all the images to the same reference image, the images with the greatest focus difference will generally have the least similarity to the reference image and as such will be harder to align, it is also likely that they have the greatest displacement as the image drift in a focal series is typically linear, for the MI method the time taken scales with the assumed maximum displacement so large displacements are best avoided. Registering only adjacent images will usually ensure the image pairs being registered are as similar as possible and that the displacements are also small which will aid the registration process. A potential downside of registering only adjacent images however is that the displacements of the images far from the reference image will suffer from an accumulation of errors from summing all the previous displacement measurements to calculate the displacement from the reference image, this means the images far from the reference are likely to be registered less accurately than those images close to the reference image just as in the previous method. Additionally, if one of the images in a series is of very poor quality, the potentially inaccurate displacement estimate

from this image will also have a knock-on effect on all subsequent measurements that include this displacement, using this method of alignment, poor images must be identified and removed from the series in order to not affect the other image displacement measurements.

There are several schemes to try to lessen the effects of these sources of error. The approach taken in FTSR [Meyer et al. 2002] which is replicated here, is to pick a reference image, and to register the two nearest images to the reference image using the PCPCF technique. At this point with 3 images aligned it is possible to perform EWR on the 3 image stack. From the preliminary 3 image reconstruction a predicted image is generated at the approximate focal plane of the as yet unregistered 4th image by the method described in section 2.3.2.4. The 4th image is then registered to the predicted 4th image from the reconstruction instead of to any of the actual images in the series. This process is repeated until all of the images are aligned to the reconstruction with each image being included within the reconstruction for the next iteration after it is registered. The benefit of this is that the predicted image should be very similar to the image to which it is being registered which aids that alignment routine, the noise in the predicted image should also decrease as the number of images in the reconstruction is increased. The image displacements will also not be affected as much by one bad displacement estimate from a low quality image as its effect on the predicted image will be limited if all the other images are correctly aligned, there should also be no accumulation of errors for images far from the reference as they are all being effectively aligned to the same image.

A second optional step can be performed after the full series has been aligned in which a predicted image can be generated for each image in the series using the data from *every other* image in reconstruction, the image can then subsequently be realigned to the reconstruction now that a more complete dataset has been used in generating the predicted images. If there are significant changes to the positions of any of the images in the series then the process can be repeated additional times until the image displacements converge. This is similar to the approach used in many of the iterative EWR schemes like maximum-likelihood (MAL) or IWFR in which the image alignments are refined against the predictions periodically throughout the reconstruction process until the positions have converged [Coene et al. 1996; Allen et al. 2004].

Another alternative approach is a combination of the first two methods in which the image displacement is not just measured between each image and either an adjacent image or the reference image, but rather is measured between each image and as many of the other images as it is possible to get a good estimate of the displacement from. This will give us multiple estimates of the displacement between each image which can then be use to generate an average displacement using a linear

least squares minimisation. It is also possible to weight the different displacement measurements according to the quality of the correlation between the images to further improve the displacement estimates.

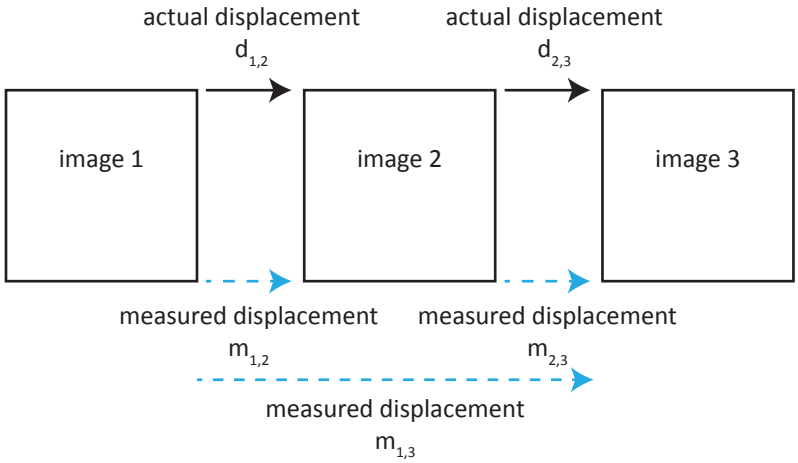

Figure 4.8: The image displacements can be overdetermined by measuring the displacements between adjacent images and also between non adjacent images. By solving a system of linear equations, the displacements can be found from a potentially much larger number of measurements which should increase their accuracy.

For a simple example system of just 3 images as shown in fig. 4.8, where the displacements are measured for all possible image pairs ($1 \rightarrow 2$, $1 \rightarrow 3$, $2 \rightarrow 3$), the matrix formulation can be written as in eq. (4.10). For larger series with a greater number of measurements the overdetermined system of linear equations can be solved using singular value decomposition (SVD) or other linear equation solving methods to find the best estimate of the image displacements. Here $d_{i,j}$ is the actual displacement between image i and j and $m_{i,j}$ is the measured displacement between image i and j from one of the procedures previously described.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} d_{1,2} \\ d_{2,3} \end{bmatrix} = \begin{bmatrix} m_{1,2} \\ m_{2,3} \\ m_{1,3} \end{bmatrix} \tag{4.10}$$

4.4 Defocus Calibration

In order to take a focal series with accurate control over the focal step between the images it is necessary to calibrate the size of the focus change resulting from a specified change in the lens DAC value. This allows experiments to be performed in an optimal manner where the optimum focal step for series acquisition is known prior to image acquisition and the microscope configured to image under these optimum

conditions [Buist et al. 1996a]. To calibrate the focal increment of the microscope it is necessary to determine the current level of focus in the microscope, and then to offset the defocus DAC value by a specified increment and the redetermine the new focus (see section 2.5.3). The primary method used to determine the level of focus is to take an image of an amorphous sample, and to determine the focus from the positions of the Thon rings in the diffractogram [Vulović et al. 2012].

In an aberration corrected microscope, measurement of the current focus level can usually be achieved using the software provided for alignment of the TEM imaging corrector as this is an integral part of the aberration determination procedure. Measurement of imaging corrector aberrations is usually achieved by determining the focus and astigmatism level for varying beam tilts by means of matching experimental diffractograms to simulated templates, this can also be done to determine just the focus and astigmatism levels without any induced tilt, if this is done for several different defocus DAC values then the difference in determined focus can be used to calibrate the focus change for image series acquisition.

As the calibration is a necessary part of the reconstruction process, a simple diffractogram matching routine has also been developed and implemented within this EWR plugin to allow for an accurate calibration of the focal step on microscopes that do not already provide this functionality through an aberration corrector or otherwise. This works by simulating the diffractogram in a heavily data reduced form for a large number of possible imaging conditions (defocus df, astigmatism magnitude $|A|$, and astigmatism angle $\arg(A)$, with spherical aberration C_s fixed). The simulated diffractograms are then compared against the original diffractogram by means of a normalized cross correlation (NCC) to find the set of image conditions which produce the best match to the experimental diffractogram. The diffractogram intensity as a function of spatial frequency $D(k)$ for a given set of imaging conditions is approximated by eq. (4.11) which ignores the effects of noise and the damping envelopes caused by temporal and spatial coherence to allow for a fast calculation as a large number of trial diffractograms are to be calculated. The experimental and simulated diffractograms are reduced down to a size of just 128×128 pixels to allow for the simulated diffractograms to be produced very quickly as a large number have to be simulated due to the dependence on 3 different variables. The diffractograms are truncated at a specified maximum spatial frequency to prevent matching noisy data towards the information limit of the microscope, this also prevents wasting the limited amount of available data on higher spatial frequencies that may contain very

little information.

$$D(\mathbf{k}) = \left(-\sin\left[2\pi \left(\frac{\lambda \cdot df \cdot \mathbf{k}^2}{2} + \frac{\lambda}{2} |A| \mathbf{k}^2 \cos\left(2 \arctan\left(k_y, k_x\right) - \arg\left(A\right)\right)\right)\right]\right)^2 \tag{4.11}$$

More details on the specific software implementation of the diffractogram matching routine and examples are given in section 4.6.1.3.

4.5 Phase Reconstruction

Once the image displacements have all been determined the exit wave can be calculated using the FTSR algorithm as described in section 2.3.2. This is a straightforward process in which the exit wave is formed as a sum of the Fourier transforms of each of the input images using a Weiner filter approach where the filters for each image depend on the microscope aberrations which should be determined beforehand. The initial FTSR procedure only has knowledge of the focus difference between each image and not the absolute focus level of the images which is generally unknown, it also assumes that the astigmatism is zero. Once the initial reconstruction has been performed, the preliminary exit wave itself can be used to determine the values of any residual defocus and astigmatism which can be further corrected in a second reconstruction procedure as described in section 2.3.2.2. This leaves the final result which is the exit plane wave free from any of the determined aberrations.

4.5.1 Choice of Algorithm

As mentioned in section 2.3 there are numerous possible methods to combine the information in a focal series to determine the exit wave. FTSR was the chosen method for this plugin for a number of reasons, firstly FTSR is perhaps the least computationally demanding of the available methods, all techniques require the initial alignment of the focal series, but after this step FTSR comprises of little more than several Fourier transforms for each image, other methods such as IWFR are iterative schemes which require running the entire EWR process multiple times to converge upon a result. MAL is another iterative method but in this method the inclusion of non-linear imaging effects into the formalisation increases the computational burden significantly for each iteration.

The computational difficulty was an important aspect when selecting a reconstruction algorithm for this plugin as one of the goals of writing this plugin was to develop EWR as a technique that could be routinely applied and deliver results in under a couple of minutes rather than requiring extensive offline processing. This

plugin is designed to be used at the microscope, and require as little user input as possible to produce a reconstruction.

The ability to produce results so quickly allows for any issues which are only encountered upon performing the reconstruction to be rectified immediately, i.e. if another series needs to be recorded it is useful to be aware of this when you are still at the microscope looking at the same area. To further facilitate this, the reconstruction process runs entirely in the background within DM so the microscope and software can still continue to be used as normal even while a reconstruction is still being performed.

4.6 Parallelisation

The process of aligning a large series of images (often 20 or more) can be very time consuming, even using the relatively fast cross correlation based methods. In the case of EWR where it is often necessary to determine the defocus difference between images utilising PCPCF, the cross correlation will be performed multiple times for each image in the series further compounding the problem. Alternatively if the MI method is required for image alignment the procedure can take substantially longer, particularly in the case of substantial misalignment between images where a great number of trial MI values are calculated for each image alignment. The trend towards larger imaging sensors providing even greater amounts of data and faster sensors allowing for even longer image series only exacerbate this problem highlighting the need for image alignment routines and general image processing software which is able to take advantage of modern highly parallel computing architectures to keep up with the computational demands.

Luckily many of the computational problems dealt with in the FTSR algorithm can be described as 'embarrassingly parallel', this term describes problems which can be subdivided easily into many smaller tasks, with little to no dependence between the subtasks so that they can be completed in isolation. In the FTSR algorithm many operations which act over an entire image can be separated on a individual-pixel basis where the calculation for each pixel has no dependency on the calculation for the other pixels. Essentially, with enough processors, every pixel of these sub routines within the full procedure could be calculated simultaneously.

This class of calculation can be easily parallelised and can often see great performance increases when implemented on GPU hardware which feature large numbers of processors. GPUs offers enormous computational performance but usually require the problem to be divided into several thousand subtasks in order to see any benefit, they can also struggle when dealing with complicated problems that require communication between the individual subtasks.

Additionally many of the standard algorithms and functions required in the FTSR procedure (such as Fourier transforms, linear algebra etc.) already have high performance implementations developed by the GPU hardware manufacturers which can easily be incorporated into the plugin to significantly reduce the effort required to rewrite the entire algorithm for GPUs.[1]

For this EWR plugin, the parallelisation was incorporated under the framework of OpenCL, described in section 1.4.3.2. OpenCL supports parallelisation on both CPUs and GPUs without requiring specific changes in the underlying code, and although it is certainly possible to optimise for specific hardware this should not be necessary. OpenCL is also compatible with both of the popular GPU manufacturers so the plugin will be able to provide GPU parallelisation on any relatively modern GPU rather than using CUDA which would limit its use to only nVidia hardware. On systems without a dedicated modern GPU the software will fall back to being parallelised on a multi-core CPU which is suboptimal but still considerably better than serial operation on any modern computer.

Parallelising this code allows for a substantial decrease in the time taken to perform EWR on a focal series over serial implementations of the same algorithm, FTSR has currently been implemented in a commercial software package licensed by HREM Research Inc. which provides a similar feature set to this plugin for performing EWR built into DM via the FTSR method. A typical reconstruction using this commercial software can take up to 45 minutes (not including image acquisition). It will be shown in section 4.6.1 that with the benefits of GPU computing, the time taken to determine the exit wave using FTSR can be reduced to well under 1 minute. In cases where the defocus does not have to be determined from the image series, only the translational image displacements, the time can be lowered to around $2s$. Using the latest generation of GPU hardware for the computation, or utilising multiple GPU parallelisation which has not been considered here, this could possibly be further reduced to sub $1s$. This is very close to the point at which it would be possible to implement a real-time EWR system capable of calculating the results of a reconstruction at the same time as the images are being acquired so the results are available almost immediately after acquisition of the final image. With faster frame rate cameras a system could be designed in which the microscope focus is continually varying on a closed loop and a reconstruction is being continuously performed in real-time over a moving series of the last few images, this could yield an EWR with the same frame-rate as the CCD acquisition.

[1] cuFFT and cuBLAS are available for nVidia hardware using CUDA, and clMath (clFFT and clBLAS) are available from AMD for OpenCL.

4.6.1 Optimisation for Graphics Processing Unit Architecture

The EWR plugin was developed in C++ using the freely available SDK for DM provided by Gatan. Although it is possible to develop scripts for DM using its built in scripting language, it does not offer to the same performance or flexibility as developing in C++ using the SDK due to limitations in the way the code is executed. It is also not possible to combine the built-in scripting language with the GPU computing features like CUDA or OpenCL, use of these requires additional libraries which can only be included when compiling using any of the major programming languages like C++,Python,Fortran etc. Both the SDK and DM itself are freely available from Gatan.

4.6.1.1 Image Alignment - PCPCF

As described earlier, the full image alignment routine using PCPCF alignment aligns the first two images using PCPCF, subsequent images are aligned by performing exit wave reconstruction on the previously aligned images to generate a predicted image with which to perform PCF. This procedure is described in the pseudocode in alg. 2.

The actual determination of the PCPCF and PCF from the two images can be divided into 4 separate sections. The first is computation of the forward Fourier transforms of the two images to be registered. The second is calculation of the phase compensated cross-correlation from the two Fourier transforms, then finally the cross-correlation is inverse Fourier transformed which leaves a map of the correlation at different displacements. After this point the position of the maximum value within the resulting image is determined, and then refined by the 3-point parabola method described in fig. 4.5 to find the final image displacement with subpixel precision.

The forward and inverse Fourier Transforms are performed in OpenCL using the clMath library from Advanced Micro Devices, Inc. This library is already optimised for performing FFTs in parallel on GPU hardware and will greatly outperform most FFT implementations that do not leverage GPU computing unless the images to be transformed are very small. The disparity for small image sizes is mostly due to significant overheads in uploading images into GPU memory and preparing the GPU kernel for execution that can take longer than the actual computation for very small images and are unnecessary when then computation is not performed on the GPU.

The search method to find the location and of the maximum value within the final image can also be efficiently parallelised on GPUs where the maximum of small subdivided regions of the image can each be found in parallel, then the same technique can be applied recursively to find the maximum of the new smaller subset of maxima again in parallel. Eventually as the size of the pool of numbers to be searched through is reduced, it reaches a point where it is more efficient to find

the maximum of the remaining numbers on the CPU instead due to overheads in running functions on the GPU. This is shown pictorially in fig. 4.9.

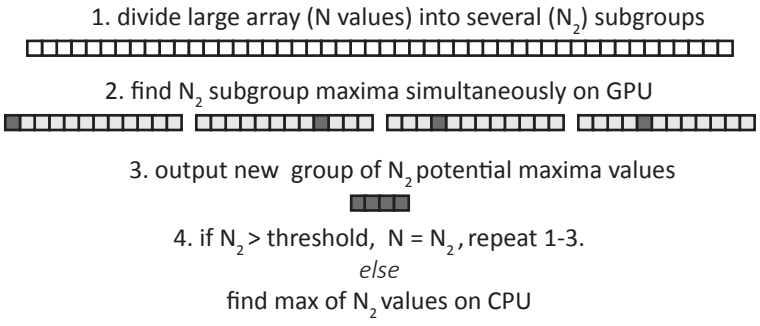

Figure 4.9: Finding the maxima of a large array of numbers can be parallelised by dividing the array into many smaller groups of numbers, and then finding the maxima of each subgroup of numbers in parallel. Then in a second step we can find the maximal value from all the potential maxima found in the first stage. This technique can be applied recursively until there are only a few maxima possibilities at which point it is faster to search through the small amount of numbers remaining using the CPU.

The calculation of the actual cross correlation from the Fourier transforms is calculated by the means of a custom written GPU kernel which can be seen in full in appendix A.3.1. This allows for a parallel calculation of each individual pixel in the PCPCF, the spatial frequency represented at each pixel is also pre-calculated on the CPU and delivered to the kernel through global GPU memory to reduce the amount of calculation required in the kernel as these values are reused on numerous occasions.

4.6.1.2 Image Alignment - MI

The implementation of MI on the GPU is more involved than for the Fourier based registration methods. MI calculation relies on the generation of multiple histograms from the image data, this is something which is not typically well suited to be computed on a GPU although there are still several semi-parallel methods which can perform well in specific conditions. Histograms are typically inefficient to be computed on a GPU for a variety of reasons, the first of these is the random memory access pattern, one thread may be writing to one histogram bin, while an adjacent thread from the workgroup is writing to a completely different bin as the write location is only dependent on the pixel intensities, this prevents memory access coalescing where many small serialized write operations are grouped into one larger write operation which is only possible when adjacent processors are reading and writing from adjacent memory locations. The second major problem is the presence

of 'race conditions' where the ordering of a sequence of events can influence the result without careful synchronisation. This is typically a problem when you have 2 or more parallel threads that are both trying to read from and then modify the result at the same location in memory, if one thread reads, modifies, and then writes before the other thread begins reading then everything works as intended, if however both threads read the value before either of them modify and write a new value back, then the first change will be written back and subsequently overwritten by the second change as if it had never occurred. This problem can be solved by using a type of variable called atomics. Atomics are locked when one thread tries to access them so that any other thread must wait until the lock is released before it can act, this prevents multiple actions occurring at the same time at the cost of slower access to the atomic variables. Further improvements can be made by having each workgroup maintain its own personal copy of the histogram (a partial histogram) stored in shared memory (memory which is much faster to access but less abundant). Each workgroup then only writes updates to its own partial histogram where there is much less of a problem with memory bandwidth and less threads with which to clash, and then at the end all partial histograms are combined to give the final result. This second method can be hard to implement in the case of a joint histogram however because of the increased number of bins in a 2D histogram, shared memory is only available in limited quantity and this not large enough to store a 2D histogram of 256×256 bins on current generation hardware.

The first problem could potentially be resolved by parallelising the kernel in an alternative manner. The standard approach would be to launch a single thread for each pixel in the image, use the thread to determine which bin to increment, and then increment the bin using atomic operations to prevent errors. Alternatively a single thread could be launched for each of the possible 256 by 256 histogram bins, each thread would then look at every pixel and only increment its own bin if it finds a matching value. This would ensure there are no write conflicts or race conditions as each thread writes to a different location. It would also allow for an optimised memory access pattern because images could still be read into shared memory in a coalesced manner and then accessed in shared memory by the threads in the workgroup. The downsides of this approach are that each pixel of the image has to be read in once for each of the workgroups rather than once in total, and also each thread has to check every single pixel rather than just one pixel. This means that whilst the second approach would have optimal memory access patterns and no write conflicts, the massive increase in the number of comparisons that would be made (65536 times more for a 256*256 bin histogram) would make this approach far slower.

The approach taken in this plugin is slightly different from the approaches

mentioned so far whilst implementing some of the same ideas. The image alignment procedure in this plugin for MI consists of performing the same joint histogram calculation multiple times with the same two images, however a small offset is applied to the second image each time to calculate MI for slightly different image displacements. When the MI is evaluated inside the GPU kernel, each thread determines the histogram bin for the first image, and the histogram bin for second image for a specific pixel, the only difference when the image is offset by a single pixel is that each thread is then calculates the histogram bin for the second image that was already determined by an adjacent thread for the previous image displacement. This allows the MI calculation to be optimised by saving the calculated histogram bins into shared memory so that the other threads in the workgroup can read the histogram bins for other image displacements from the shared memory rather than recalculating them in a separate MI calculation which would require reading the image intensities from global memory again.

Because of the limited size of the workgroups, pixels at the edge of the workgroup will require histogram bin values for pixels that are calculated as part of an adjacent workgroup. Because shared memory is only accessible within the individual workgroups, each workgroup also has to calculate the bins for a small overlapping area around one edge equal to the number of join histogram displacements being simultaneously calculated as these histogram bin values would otherwise have been calculated in a different workgroup. This method also has some other limitations, the number of joint histograms that can be calculated at once is limited by the number of joint histograms that can be stored at once using the GPU memory, and also the number of histogram bins that can be stored at once within the shared memory of a single workgroup. Each joint histogram is 256×256 bins which is $256kb$, to calculate a 50×50 grid of possible image displacements would take $625MB$ of memory on the GPU just to store the joint histograms. For a typical workgroup size of 32×8 threads, shared memory would also have to be allocated for $(32+49) \times (8+49)$ histogram values from the offset image. The histograms also still have to be updated using atomics to prevent problems from multiple threads updating the same value. The upside is that the kernel would only need to be launched once to get all 2500 joint histograms, rather than 2500 times, and the histogram bin for each pixel would only be calculated 24 times (each pixel would be needed in 24 different workgroups due to the overlap size being greater than the workgroup size in both x and y directions), instead of 2500 times. The reduction in the number of kernel launches can be beneficial due to the overheads incurred each time a GPU kernel is launched. The full code for this MI kernel can be found in appendix A.3.2 and an overview of the MI procedure is given in alg. 3.

4.6.1.3 Diffractogram Matching

The diffractogram matching procedure included within this plugin is used for microscope defocus calibration which is necessary for the image acquisition routine, it can also be used for determining the correct focal position is achieved prior to series acquisition. The routine can be described in terms of two main sections. The first of these is the preprocessing of the original diffractogram to significantly reduce the amount of data in the diffractogram, and the second is determining the best possible imaging conditions $(df, |A|, \arg A)$ that produce a matching simulated diffractogram to the experimental one.

The preprocessing is used to scale the diffractogram down to 128×128 pixels as producing simulated diffractograms of this size was deemed optimal as a compromise between quality and time required for finding a match, both of these tasks scale linearly with the number of pixels.The diffractogram data beyond a maximum spatial frequency f_{max} in the x and y direction is also discarded as the SNR within the diffractogram decreases with increasing spatial frequency due to the limited coherence of the electron source making the higher spatial frequency data in many images unusable. Information at greater spatial frequencies is still present in corners of the image, this is not removed to ensure the processing time is as low as possible. The spacing between Thon rings becomes smaller at high frequencies which will not be represented well in a diffractogram with such a low number of pixels which further supports the discarding of higher spatial frequency data. The remaining data is then scaled such that the spatial frequency range of the processed diffractogram goes between $-f_{max}$ and f_{max} in both the x and y direction. The factor with which the diffractogram is downsampled, D_{scale} is calculated as eq. (4.12) where Nx is the number of pixels in one dimension of the original image and $f_{origmax}$ is the maximum frequency in the original diffractogram. An example of this process is given in fig. 4.10.

$$D_{scale} = \frac{f_{max} \times Nx}{64 \times f_{origmax}} \tag{4.12}$$

If D_{scale} is less than 2 then we can form the resampled diffractogram by bilinear interpolation from the original diffractogram values, however if D_{scale} is greater than 2 then bilinear interpolation is not particularly accurate as it would lead to some of the image pixel values being unused in the resampling process and so the image is first binned by a factor of 2 (a new image is formed of with dimensions equal to half of the original image where each pixel is formed from the values of a non-overlapping 2×2 square of pixels in the original image), and D_{scale} is halved recursively until it is less than 2 at which point bilinear interpolation is used to form the final diffractogram without having discarded any of the original data.

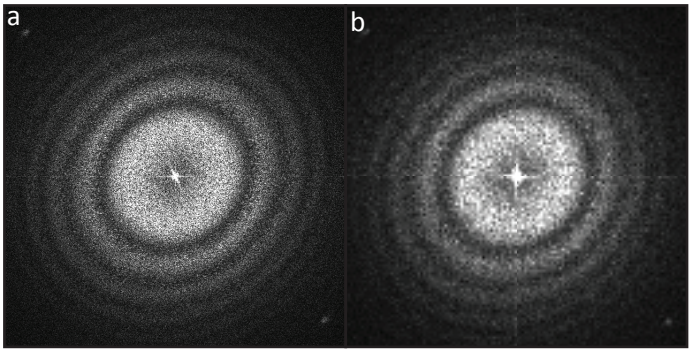

Figure 4.10: a) An original diffractogram of size 2048×2048 and b) the compressed version reduced to 128×128 with a maximum spatial frequency f_{max} of $4nm^{-1}$ used in the template matching procedure.

This processed diffractogram is then used as input for the a custom GPU kernel (full code in appendix A.4.1). The diffractogram is further split into 4 separate quadrants of 64×64 pixels, these quadrants are small enough that they can be stored in the constant memory cache of the GPU, this is highly advantageous as when multiple processors try to access the same data elements simultaneously from constant memory, only one memory read is required to serve all the processors at once, the full 128 by 128 image cannot be stored within constant memory. For this kernel the parallelisation is expressed in such a way that each thread is responsible for calculating a full simulated diffractogram quadrant for an individual set of imaging conditions whilst concurrently comparing it to the original data and returning a NCC result for its own specific set of imaging conditions. The kernel is then launched again a separate time to compare another quadrant from the original diffractogram, and the correlation coefficients for the same imaging conditions and different quadrants can be accumulated to generate the same results as if the full diffractogram was compared at once which is not possible on current hardware using this method. This process is necessary to allow for the use of constant memory which greatly enhances the performance of the kernel and the kernel only needs to be launched 4 times (once per quadrant) as opposed to a separate kernel launch for each set of imaging parameters (if we parallelised over the pixels in the diffractogram instead of over the imaging conditions). This approach drastically reduces the overhead from kernel launches when $1,048,576$ different imaging parameter combinations are tested for each kernel launch (128 defocus, 64 astigmatism magnitude, and 128 astigmatism angle).

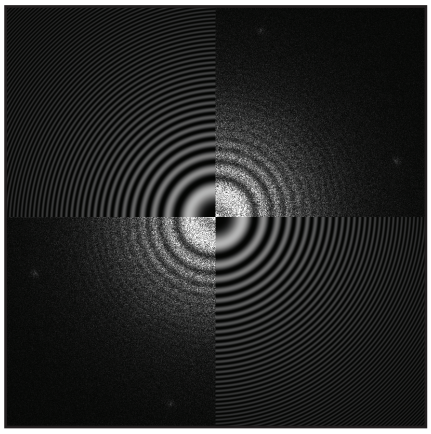

Figure 4.11: A composite image of an experimental diffractogram from a focal series image and a simulated diffractogram for the imaging conditions determined using the diffractogram matching routine described in section 4.4, in this case the best match was found for $df = -281.5nm \pm 0.125$ and $A = 9.375 + 9.375i\,nm\ \pm 0.125 + 0.0125i$

Figure 4.11 shows that the diffractogram matching routine is able to accurately match the positions of all the Thon rings within the image for an exemplary diffractogram with a large number of Thon rings and good SNR even in the cast of a slight astigmatism. This entire process takes only a few seconds using a nVidia GTX 570 for the computation. The routine returns a composite image which is part simulated diffractogram and part experimental to allow the user to easily determine the quality of the defocus and astigmatism determination.

If the defocus and astigmatism are already approximately known, the diffractogram routine can be run with a much smaller number of trial imaging conditions. In this case the execution time can be reduced to much less than 1 second, this allows the diffractogram routine to be run in real-time on the live camera feed from the electron microscope and return real time defocus and astigmatism values with a very short delay if the diffractogram is of sufficient quality.

4.6.1.4 FTSR

The calculation of the exit-wave using FTSR (explained in detail in section 2.3.2) requires the Fourier transform of each registered image c_i, and the wave transfer function w_i. The wave transfer function is derived from the imaging conditions under which the image was taken (w_{i-} is the wave transfer function for $w(-\mathbf{k})$) and is complex array of values with dimensions equal to the image. These two objects are then combined as in eq. (4.13) to generate the 5 running sums over the full image series [Meyer et al. 2002]. These 5 sums are then combined using eq. (4.14) to give

the resultant exit wave from all the images included in the sum.

$$\begin{aligned} W &= \sum_i |w_i|^2 \\ W_- &= \sum_i |w_{i-}|^2 \\ U &= \sum_i w_{i-} c_i \\ V &= \sum_i w_i w_{i-} \\ T &= \sum_i w_i^* c_i \end{aligned} \tag{4.13}$$

$$\Psi_{ew}(\mathbf{k}) = \frac{(W_- + \nu)T - V^* U}{(W_- + \nu)(W + \nu) - |V|^2} \tag{4.14}$$

This part of the reconstruction procedure is straightforward to implement into the plugin via GPU kernels which can be used calculate the 2 wave transfer functions w_i and w_{i-} directly in GPU memory from the determined imaging conditions. In these kernels the parallelism is expressed by launching a separate thread for every pixel in the image, which can all be determined independently. A separate GPU kernel has also been written to perform the calculation of each of the 5 running sums in eq. (4.13), the sums are performed directly on the GPU to minimise the unnecessary transfer of data between GPU and host memory which can be time consuming. These GPU kernels can all be found in appendix A.2.

The final part of the FTSR procedure is to correct any residual defocus and astigmatism leftover in the reconstructed exit wave. This process is done using the PCI approach of Meyer et al. detailed in section 2.3.2.2. This section of the reconstruction process is one of the most time consuming, its performance has been improved considerably through the use of GPU computing. The phase contrast index eq. (2.23) is determined for a given exit wave for a trial defocus and astigmatism value, it's value should be positive for all spatial frequencies if these chosen trial values match the actual values for the exit wave. By searching over a range of trial values and finding the maximum integrated f_{pci} we can locate the correct defocus and astigmatism parameters.

The f_{pci} calculation is performed in a custom GPU kernel which calculates the f_{pci} parallelised over each pixel in the image, and then another GPU kernel which calculates the sum of the calculated f_{pci} values for every pixel via a parallel sum reduction method. The parallel sum reduction to sum a total of N numbers, uses W workgroups which together have a total of X threads with W«X«N, each thread then starts from its own id ($0 \rightarrow X-1$), and adds this pixel value to its own running

total, and then skips forward by X values to the next number to be summed. After all of the values have been processed, the second stage starts where the first thread of each workgroup adds up all the running totals from each of the $\frac{X}{W}$ threads within its own workgroup. This leaves just a small number of values, W, remaining to be summed. These final few values are copied back to the host computer and summed on the CPU, as the GPU will perform slower than the CPU for calculating the sum of a relatively small amount of numbers. The two kernels are shown in appendix A.2 and appendix A.4.2. This type of task can be very computationally intensive as for every trial defocus that is tested, the PCI calculation involves calculating an array of numbers with the same dimensions as the original image from the exit wave, and then performing a summation over every value in this array. If the approximate defocus for the reference image is not known with reasonable accuracy prior to reconstruction then a large number of trial values may need to be searched over.

4.6.2 Performance Improvement

The effect of exploiting the inherent parallelism in the reconstruction procedure on either multicore CPUs or GPUs is to drastically reduce to time taken to perform the full exit wave reconstruction procedure. The results in fig. 4.12 and fig. 4.14 show the time taken to align two images via the PCPCF procedure and the time taken to perform the entire reconstruction for a 20 image series respectively. Figure 4.13 shows the time taken to add a single aligned image to the reconstruction as this is performed numerous times during the overall procedure making up a large percentage of the total time. For this comparison two different GPUs have been used, a Nvidia GTX570 with 480 processors at 732 MHz, and a AMD Radeon HD7950 with 1792 processors at 950MHz, however the number and speed of processors does not perfectly reflect the performance due to different architectures involved. The CPU used in the comparison is a Intel Core-i7 2600K with 4 cores operating at 3.4GHz. The plugin has been benchmarked at a range of different resolutions from 256^2 up to 2048^2 as the performance of GPU is more likely to fluctuate with the size of the workload than for a CPU.

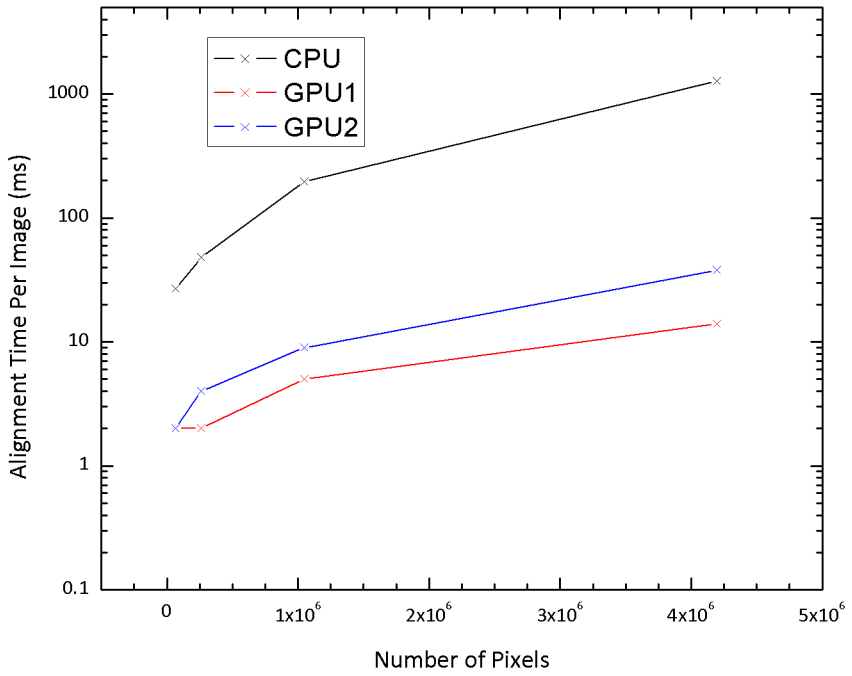

Figure 4.12: Performance profiling for the time taken to perform the image alignment procedure for a single pair of images via PCPCF using different resolutions for both CPUs and GPUs. The results are shown on a logarithmic scale to improve visibility for the GPU times.

In the case of single image registration, the performance difference between a multicore CPU (4 processors) and a GPU (480 processors) ranges between 20× faster and 30× faster for the GPU as the resolution is increased from 256×256 to 2048×2048 pixels. The faster GPU however is over 90 times faster at the largest resolution where a single image registration takes ~ 1.3 seconds for the CPU and just 14ms on the GPU. Clearly the performance of the CPU is unsuitable for performing registration of high resolution images at the fast frame rates that would be necessary for a real time reconstruction procedure whereas both GPUs can register multiple images a second. The performance advantage of the GPUs is more prominent for larger images as increased workload can be used to amortize the inherent latency of the GPU and the overheads contribute less to the total time.

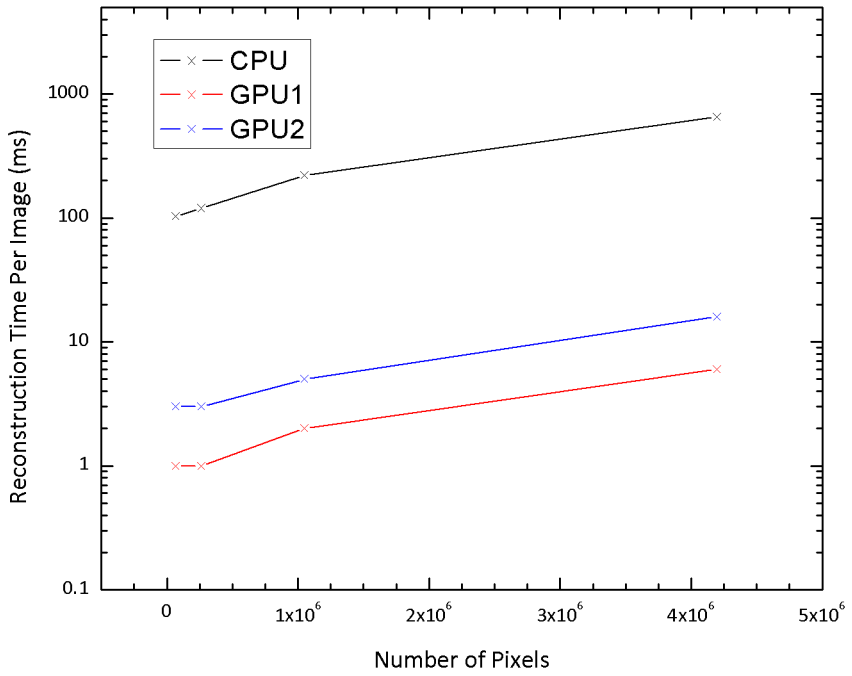

Figure 4.13: Performance profiling for the time taken to add an aligned image to the reconstruction using different resolutions for both CPUs and GPUs. The results are shown on a logarithmic scale to improve visibility for the GPU times.

Figure 4.14 looks at the time taken to perform the entire reconstruction procedure on both CPU and GPUs again at a range of different resolutions. Here, the performance increase demonstrated for single image registration is reflected in the overall total reconstruction time, the slower GPU is again $\sim 20\times$ faster at a resolution of 256×256 pixels which increases to just over $30\times$ as the resolution is increased to 2048×2048 pixels, the faster GPU is just over 75 times faster at the largest resolution tested. A reconstruction that takes over 5 minutes on the CPU is performed in under 5 seconds on the GPU. The performance at resolutions of 1024×1024 pixels and below already shows promise for being able to perform real time exit wave reconstruction, a 20 image series can be reconstructed in $0.7, 0.8$ and 1.8 seconds respectively on the higher performance GPU, this can be expressed as frame rates of $28, 24$ and 11FPS at which a live reconstructed image could be displayed assuming a microscope capable of acquiring differently defocused images at an equal rate.

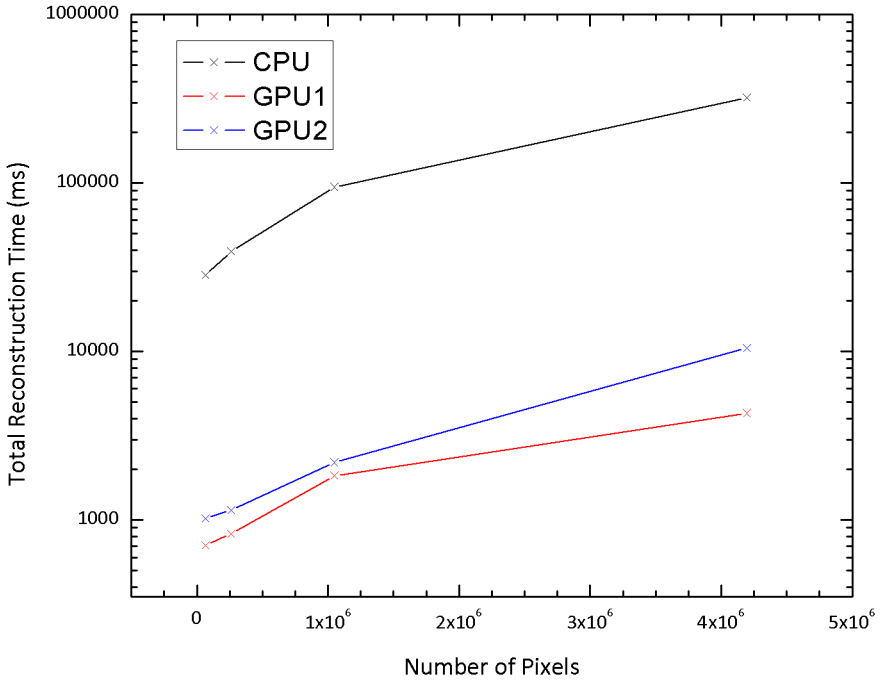

Figure 4.14: Performance profiling for the time taken to perform a complete exit wave reconstruction procedure for a 20 image series using different resolutions on both CPUs and GPUs. The results are shown on a logarithmic scale to improve visibility for the GPU times.

The majority of the reconstruction time is taken up by adding the individual images to the reconstruction after they have been aligned. This step is performed for every previously aligned image each time a new image is registered as the images are themselves aligned to a restored image for improved accuracy, this means the cost of adding an image to the reconstruction is incurred $\sum_{i}^{N} i$ rather than just N times. If images were just aligned to a neighbouring image without requiring the additional reconstruction steps the reconstruction time could be reduced to N times the single registration time plus N times the reconstruction addition time plus the fixed time taken to determine the focus and astigmatism, this would further increase the reconstruction speed if necessary. If the defocus/astigmatism determination were not needed (i.e. the reconstruction is performed at a chosen image plane with the current level of astigmatism), the reconstruction for even 2048×2048 pixel images could be performed at above 20FPS based on the time taken to register and add a single image to the reconstruction shown in fig. 4.12 and fig. 4.13. This particular algorithm can also be easily adapted to operate on a moving sequence of images (i.e.

performing a reconstruction from the last N images acquired from a live view), each time a new image is acquired, to update the previous reconstruction we only need to performance 1 alignment, 1 image addition, and 1 image removal (removal takes the same length of time as addition), this should take approximately 26ms total on the faster GPU which could allow for operation at nearly 40FPS.

4.6.3 Integration into Digital Micrograph™

There are two possible ways to develop plugins for DM, DM provides its own scripting language which includes a reasonably complete suite of inbuilt functions to perform most simple data processing tasks as well as enough functionality to write almost any other features that would be desired yourself from the basic features provided. This scripting language also includes functions designed to interface with the microscope and receive operating parameters such as the voltage and image acquisition settings etc. The built in scripting language differs from many traditional programming languages however in that the script is not compiled into an executable before being run, rather it is interpreted on a line by line basis when it is run instead, this tends to offer less performance than generating compiled code which can be extensively optimised. The main drawback however to using this scripting language is that is does not allow for any interoperability between the vast array of specialised libraries for data processing and enhanced functionality like CUDA and OpenCL which are written for languages such as C, C++ and FORTRAN etc. As a result plugins developed using the built in language would have to manually write many specialised functions such as the vast array of matrix operations which are provided by BLAS and FFT libraries in C++, this is very labour intensive and unlikely to offer the performance of the highly tuned libraries already available. Additionally some other functionality (particularly GPU computing) is unavailable altogether without access to their libraries.

DM also provides a software development kit (SDK) which allows for plugins to be written natively in C++, this means that almost anything that can be done in C++ can also be done in a DM script, thus opening up many new possibilities. The SDK also provides an identical copy of the majority of the functionality from the DM scripting language which can simplify many of the tasks such as image acquisition and also allows for relatively simple access to elements of the DM GUI and control over displaying images. It is also possible to call scripts written in the DM scripting language from C++ and vice-versa should this be necessary. Because of the very flexible nature of writing natively in C++, it is also possible to make use of some of the more advanced features of the language to create plugins that are able to make use of multiple threads to either operate faster, or that can run intensive tasks in the background while maintaining responsive access to the DM GUI to

continue working or display progress updates. Software can also be designed which runs external (either locally or otherwise) to DM but can still exchange information with DM through a separate plugin. For the purpose of this EWR plugin it was necessary to use the SDK to be able to incorporate some of the advanced features of the code such as GPU parallelisation and multi-threading. A small part of the image acquisition routine was written in the built in scripting language and is called from C++ as some required functionality was unavailable in the SDK.

This plugin has been designed to provide a GUI that can be docked to the side of the DM interface much like most of the built in DM functionality. The GUI has dialog elements that are used for entry of the parameters relevant to image acquisition such as focal step and number of images in the series and option buttons to change the image acquisition method for ease of use. The defocus calibration parameters for the microscope are maintained within DMs own global tag system so that once they have been entered once they will remain calibrated even after the program has been closed or restarted. The dialog is split into separate tabs for the reconstruction and image acquisition sections as seen in fig. 4.15. All verbose output from the plugin is given in the DM progress window which is updated throughout the operation of the plugin, or in the DM results windows for longer messages. Details about the image acquisition conditions for a focal series are stored within tags attached to each individual image that can be retrieved whenever the images are opened later and are used to automatically set the correct parameters for the reconstruction process when the images are loaded. The plugin can also be used to restore focal series that have not been acquired using its own acquisition routine if the relevant information is added to the image tag structure. The plugin should also automatically find an appropriate device to use for the calculations (AMD or nVidia for GPUs or Intel or AMD CPU) but a specific computation device can also be chosen if more than one are available.

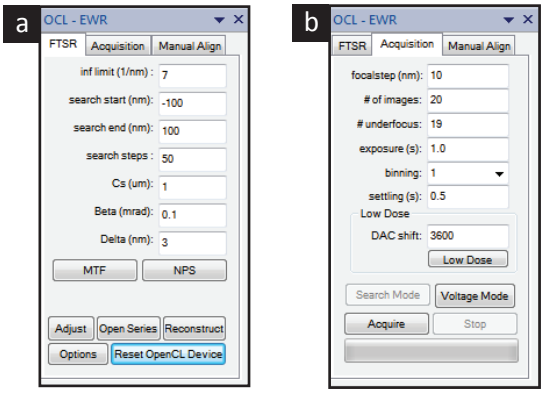

Figure 4.15: Graphical user interface for the EWR plugin in DM. a) reconstruction tab and b) image acquisition tab. These tabs are hosted within a small window which can also be docked to the side of the DM program.

When a reconstruction is performed the plugin produces numerous different outputs, these include graphs of the image drift (x and y) against the image acquisition order, a graph of the estimated image defocus against the image acquisition order, one exit wave prior to residual defocus and astigmatism correction and one exit wave after residual defocus and astigmatism correction. The plugin also outputs the entire image series after correction for any misalignment, this can be used to visually inspect the quality of the alignment routine and check if any of the images were unable to be accurately registered. The plugin also includes a tool to simulate an image with intentionally applied aberrations starting from an exit wave generated using this plugin. If the image registration was performed using MI, the plugin also outputs map of the MI against image displacement for each image.

In the following chapters this plugin will be used for the acquisition and reconstruction of focal series for various kinds of samples including amphiphilic block copolymer self assemblies and both pristine and fluorinated graphene which will make signification use of the MI image alignment technique and ability to correctly align image series with significant changes in rotation and magnification.

4.7 Chapter Summary

A high performance EWR plugin has been created for DM that uses the FTSR method to determine the exit wave from a focal series of images. The plugin also contains functionality for acquiring the image series in a heavily customizable manner as well as some additional tools relevant to the EWR process such as the diffractogram fitting routine which can be used for microscope calibration. Additional methods of image registration which have been adapted from other fields of research such

as medical imaging have also been incorporated which can offer improved image registration in regimes where the typical methods may be prone to failure. The MI image alignment method implemented here is capable of accurately aligning images where the cross correlation based methods can be prone to errors. This plugin provides high performance EWR on any modern computer, and on computers with a dedicated GPU can process a full image series in significantly under a minute offering large performance gains over current EWR software. The performance of this EWR plugin is sufficient to achieve reconstruction speeds comparable with image acquisition framerates for typical microscope CCD cameras which demonstrates the possibility to perform real time exit wave reconstruction for high resolution images from the software perspective.

Algorithm 2: Image Alignment Procedure using PCPCF

```
set reference image;
correct image series for magnification and rotation using assumed
defocus;
for image = 0 to number of images - 1 do
    if image < 2 then
        current image = closest image to reference that is unregistered;
        copy reference and current image to GPU;
        FFT both images on GPU;
        for trial = 1 to number of trials do
            calculate PCPCF between both images at trial defocus on
            GPU;
            determine highest correlation value on GPU;
        end
        record best trial value and corresponding displacement;
        add current image to list of registered images;
    end
    else
        current image = closest image to reference that is unregistered;
        for image = 0 to number of aligned images - 1 do
            add aligned image to reconstruction on GPU;
        end
        create predicted image from reconstruction on GPU;
        FFT current image on GPU;
        for trial = 0 to number of trials - 1 do
            calculate PCPCF between current image and predicted
            image at trial defocus on GPU;
            determine highest correlation value on GPU;
        end
        record best trial values and corresponding displacement;
        add current image to list of registered images;
    end
    for image = 0 to number of images - 1 do
        add image to reconstruction on GPU;
    end
    determine defocus and astigmatism using PCI on GPU;
    for image = 0 to number of images - 1 do
        add image to final reconstruction on GPU at correct defocus
        and astigmatism;
    end
end
```

Algorithm 3: Image Alignment Procedure using MI

```
set reference image;
correct image series for magnification and rotation using assumed
defocus;
for image = 0 to number of images - 1 do
    if image < 2 then
        current image = closest image to reference that is unregistered;
        copy reference and current image to GPU;
        calculate MI between both images for 60*60 trial
        displacements;
        find best displacement;
        add current image to list of registered images;
    end
    else
        current image = closest image to reference that remains
        unregistered;
        second image = reconstruction from all previously aligned
        images;
        calculate MI between current image and second image for
        60*60 trial displacements;
        record best trial values;
        add current image to list of registered images;
    end
    perform full reconstruction with all images;
    determine defocus and astigmatism using PCI;
    perform full reconstruction at correct defocus and astigmatism;
end
```

Chapter 5

Study of Fluorinated Graphene via Exit Wave Reconstruction

5.1 Introduction

Graphene has attracted a huge amount of scientific interest since its discovery in 2004 [Novoselov et al. 2004], this interest has been primarily as a result of its rather impressive mechanical and electronic properties which position it as an ideal candidate material for a huge range of devices such as radio frequency transistors, reinforcement in composites, supercapacitors, battery electrodes, flexible/wearable electronics and optoelectronics amongst others [Schwierz 2010; Wang et al. 2009; Stankovich et al. 2006; Bonaccorso et al. 2010; David et al. 2014; Meng et al. 2013; Avouris and Freitag 2014]. The lack of a band-gap in pristine graphene however is a fundamental problem that is preventing the adoption of graphene for logic applications where it could otherwise be an ideal candidate for the replacement of silicon [Schwierz 2010]. This lack of band-gap has led to an increased research interest in graphene derivatives formed through morphological and chemical modification such as graphene oxide, graphane and fluorographene etc. which can be engineered to provide a band gap [Hicks et al. 2013; Inagaki and Kang 2014]. Fluorographene has been predicted to exist in numerous phases [Sahin et al. 2011] and this chapter consists of a study of experimentally fluorinated CVD grown graphene using electron microscopy and EWR to determine the precise atomic configuration.

5.2 Models of Fluorinated Graphene

In contrast to graphene which is a single monolayer formed from 6 membered carbon rings arranged in a hexagonal grid, fluorinated graphene is simply graphene where each of the carbons may have a single associated fluorine atom. The extent of the

fluorination will determine just how many of the carbons will have an attached fluorine, and the precise locations of these fluorine atoms can have a major impact on the properties of the material. Fluorinated graphene can form a diverse range of structures as it is possible for the fluorine atoms to bind to either the top or bottom surface of the graphene sheet and there is the possibility to form ordered structures where the fluorine atoms are positioned in a repeating pattern [Sahin et al. 2011; Galvão et al. 2013]. Some of the possible ordered structures are detailed here.

5.2.1 Stoichiometric Fluorographene (CF)

Stoichiometric fluorographene (CF) is fully fluorinated graphene, i.e. graphene with one fluorine atom bound to every carbon atom. This is theoretically predicted to present itself in a chair configuration where the adjacent carbon atoms are decorated with fluorine atoms on alternating sides of the the graphene sheet(see fig. 5.1) [Sahin et al. 2011].

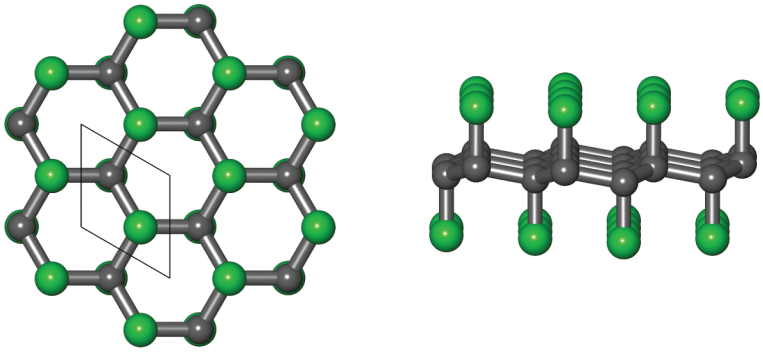

Figure 5.1: A plan and side view atomic model of stoichiometric fluorographene (CF) showing the sp^3 nature of the bonding and the alternate arrangement of fluorine atoms on both sides of the graphene sheet.

5.2.2 Chair Fluorinated Graphene (C_2F)

C_2F is partially fluorinated graphene with one fluorine for every 2 carbon atoms. This stoichiometry can exist in more than one phase, chair C_2F is a phase in which fluorine atoms are all bound to non adjacent carbons, so for every 6 membered carbon ring there will be 3 fluorine atoms positioned at the same sites (see fig. 5.2). In this configuration the fluorines are predicted to all attach to the same side of the graphene sheet [Sahin et al. 2011].

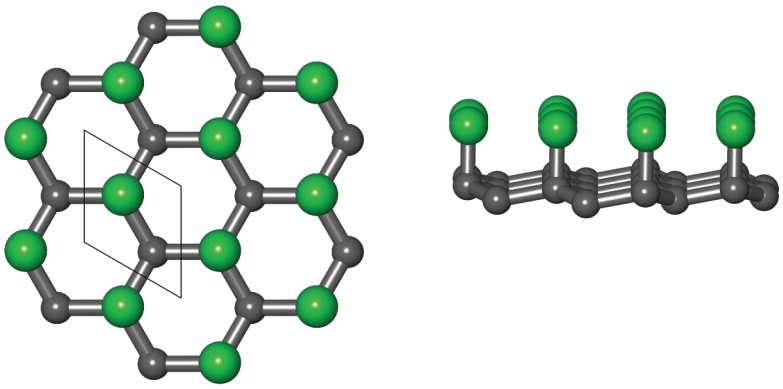

Figure 5.2: A plan and side view atomic model of chair fluorinated graphene (C_2F) showing the sp^3 nature of the bonding and the alternate arrangement of fluorine atoms on just one side of the graphene sheet.

5.2.3 Boat Fluorinated Graphene (C_2F)

There is also another phase of partially fluorinated graphene (C_2F) which loses the hexagonal symmetry altogether by placing fluorine atoms in a rectangular arrangement with alternating rows of fluorine pointing inwards and outwards from the hexagonal carbon ring (see fig. 5.3) [Sahin et al. 2011]. This form of fluorinated graphene should be easily distinguishable from the other forms of fluorinated graphene due to the departure from hexagonal symmetry which would be observable in a diffraction pattern.

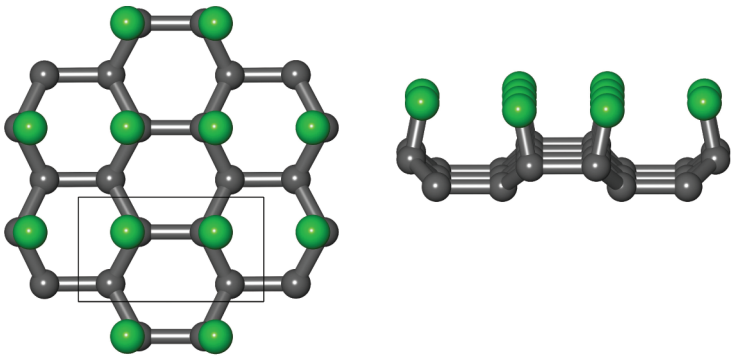

Figure 5.3: A plan and side view atomic model of boat fluorinated graphene (C_2F), the unit cell is now rectangular due to the loss of hexagonal symmetry.

5.3 Electron Diffraction Analysis

Standard HRTEM imaging is generally unable to distinguish between many of the forms of functionalized graphene and pristine graphene as the contrast produced from the various forms can be quite similar (see simulations in fig. 5.11). This means other techniques are required to determine the extent of fluorination and precise atomic configuration. Electron diffraction is a useful tool to analyse the symmetry of the sample area being imaged which can reveal a lot of information for the fluorinated graphene. The presence of boat phase C_2F in a fluorinated graphene sample would be immediately recognisable in an FFT or diffraction pattern due to the departure from 6-fold symmetry present in both pristine graphene and fluorographene as well as chair C_2F. Other ordered forms of fluorinated graphene like C_4F would also have a significantly different diffraction pattern due to the doubling in size of the unit cell required. Electron diffraction is also an accurate method to measure the lattice parameter of samples in reciprocal space which is predicted to be different for the various forms on fluorinated graphene and also different from pristine graphene itself.

5.3.1 Lattice Parameter Determination

Figure 5.4 shows overlaid electron diffraction patterns taken with near identical imaging conditions for both pristine graphene and a partially fluorinated graphene sample using an aberration corrected Jeol ARM200F Microscope at 80kV. It can clearly be seen in both the image and subsequent line profiles that the size of the lattice in the fluorinated graphene sample is larger than that of the pristine graphene (the spacing is smaller in reciprocal space). It is also evident that the 6-fold symmetry is preserved in the fluorinated graphene sample ruling out the presence of boat C_2F which does not have 6-fold symmetry and C_4F which would contain a series of hexagonal spots roughly halfway out to the first order spots in graphene.

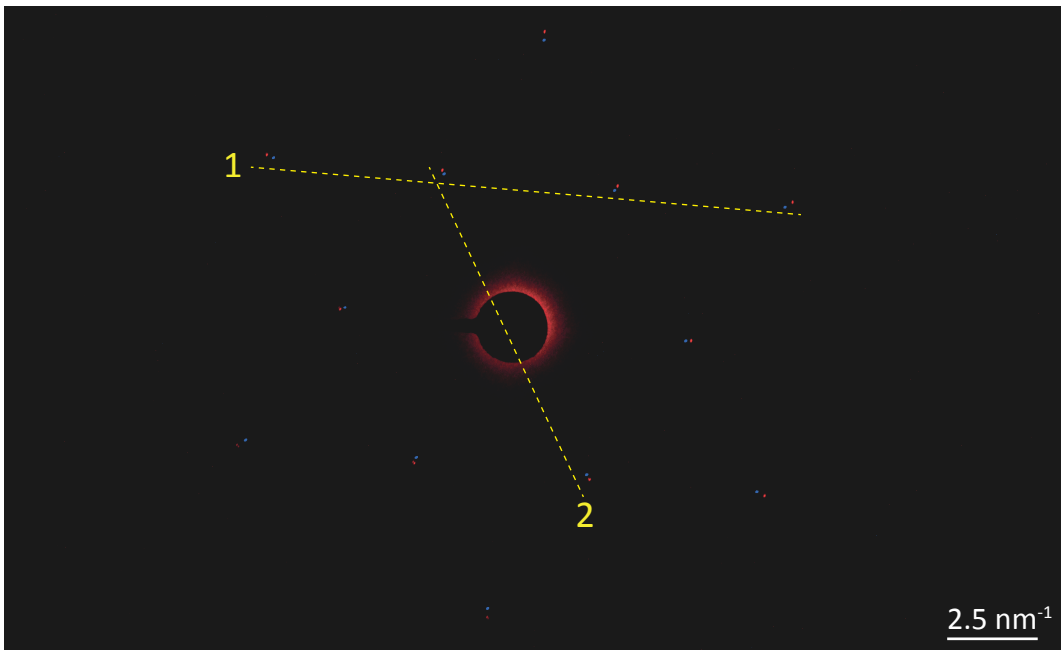

Figure 5.4: Overlaid selected area electron diffraction patterns for graphene [red] and fluorinated graphene [blue] taken under the same conditions. The yellow lines indicate the positions of the line profiles in fig. 5.5 and fig. 5.6. The graphene pattern has been manually rotated and translated to be aligned with the fluorinated graphene pattern.

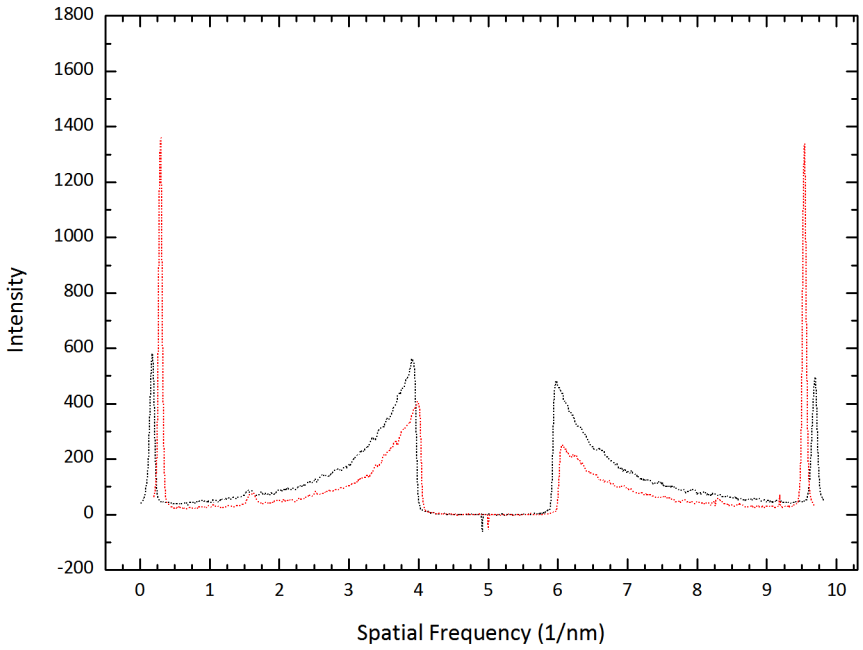

Figure 5.5: Line profile through the selected area electron diffraction patterns shown in fig. 5.4 [line #2]. The diffraction spots are further apart for the graphene [black] than for the fluorinated graphene [red] indicating that the lattice spacing of the fluorinated graphene sample is expanded with respect to that of the pristine graphene.

5.3.2 Determination of Number of Layers

The relative intensities of the $\bar{2}110$ to $\bar{1}010$ to type reflections in the hexagonal diffraction pattern can also be used to infer information about the number of layers present in the sample. In the case of pristine graphene there should be an approximate $1.1x$ intensity for the inner peaks compared to the outer peaks for a single monolayer [Wilson et al. 2010], this ratio is drastically decreased in the presence of multilayers. If a similar outcome is to be expected for fluorinated graphene, then a ratio of around 1 between the inner to outer peaks should be indicative of monolayer material. It can be seen in fig. 5.6 that the intensity ratios for the fluorinated graphene diffraction pattern are similar to those for the monolayer graphene so it is likely that this part of the sample is monolayer.

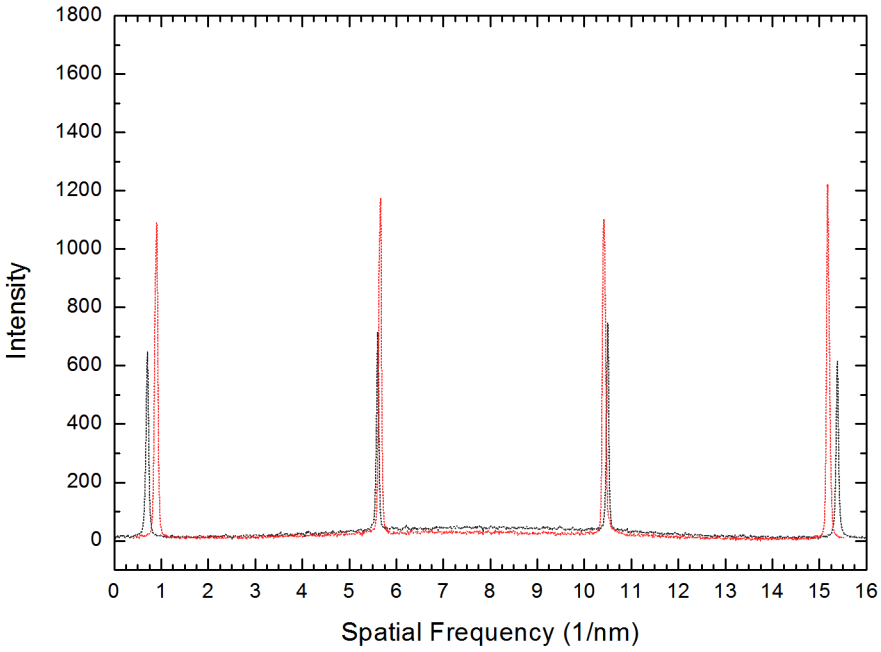

Figure 5.6: Line profile through the selected area electron diffraction patterns shown in fig. 5.4 [line #1]. The ratio of intensities between inner and outer spots is similar between the monolayer graphene [black] and the fluorinated graphene [red].

5.4 Structure Determination via Exit Wave Reconstruction

Another technique that can be used to distinguish between the various forms of fluorinated graphene is EWR, described in section 2.3. EWR is used to recover the phase change that is undergone by electrons after passing through a sample, it can also be used to remove the distortions in the exit wave caused by aberrations in the imaging optics system of the electron microscope. The phase shift at different positions in the exit wave is a result of the potential experienced by the electrons as they pass through the sample at that location.

EWR has already been used to great effect in the study of graphene where it has demonstrated the ability to clearly resolve the atomic arrangement of single and double layer graphene sheets, and even to provide 3D information on the positioning of individual atoms [Jinschek et al. 2011; Van Dyck and Chen 2012]. These experiments show the capability of EWR to clearly resolve structures down to the individual atoms and to resolve the difference in phase between a single carbon atom and a pair of carbon atoms. The presence of additional fluorine atoms at some

locations in the fluorinated graphene sample should cause an additional phase shift which can be detected in the exit wave despite having a minimal effect on individual HRTEM images. In order to determine the exit wave to sufficient accuracy to detect this phase change, a series of images must be acquired and then subsequently drift corrected to a high accuracy before the EWR process.

5.4.1 Experimental Focal Series

Image series were acquired on fluorinated graphene samples and also pristine graphene samples using the image acquisition script previously described in section 4.2 to record focal series with a nominal focal step between images of $1.5nm$. These image series were acquired on an aberration corrected Jeol ARM200F Microscope operating at 80kV. The focal step between the images was determined by a prior calibration of the microscope defocus increment (determined to be $\sim 0.75nm$) using the diffractogram matching procedure on images with an large intentional defocus change applied as detailed in section 4.4. Whilst it is possible to recover an exit wave from as few as 3 images, the quality of the reconstructions generally improves with the number of images acquired as this both lowers the noise and increases the likelihood of covering all the possible zeros within the transfer functions from the individual images. The downside to acquiring a long series is that it increases the electron dose to the sample thus increasing the likelihood of damage, and also increases the time over which the sample is allowed to drift and also the time over which the imaging conditions of the microscope can drift. Here series of 25 or more images were recorded as the sample proved to be very stable under the electron beam for extended periods of time. A focal series of fluorinated graphene, acquired using the image acquisition routine described in section 4.2, is shown in fig. 5.7.

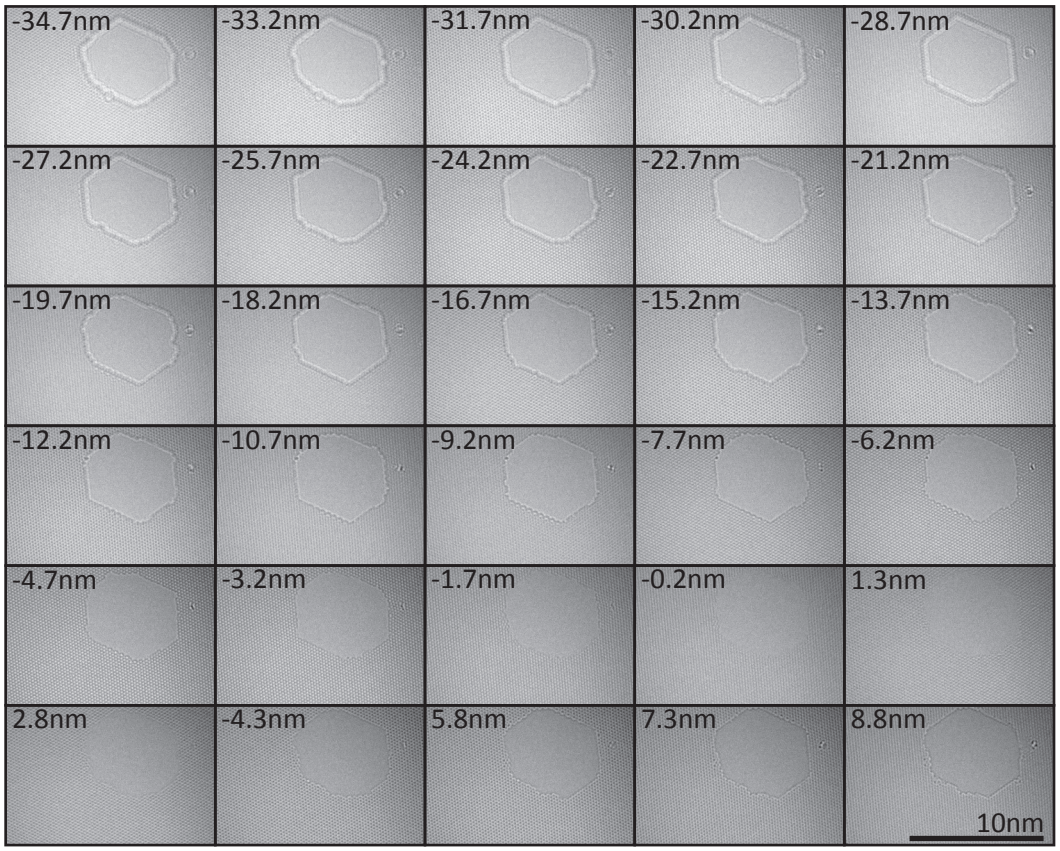

Figure 5.7: A focal series of images of a fluorinated graphene sample used for exit wave reconstruction. This series of 30 images were taken with a nominal focus increment of 1.5nm. The series is displayed in row-major order (the most under focused image is top left). This area of the sample contains a hole that was generated after a prolonged exposure to the electron beam.

5.4.2 Phase Restoration

The reconstruction process itself can be difficult for graphene like samples partly due to difficulties during the image alignment process in addition to the generally low image contrast. If the image region is purely graphene-like crystalline material, then it is hard for the alignment procedure to distinguish between several possible image alignments due to the translational symmetry of the images. The inclusion of regions of amorphous material such as polymer residue from the transfer process or structural features like holes or edges in the images can aid in the image alignment but these are often mobile during the acquisition of an image series which can also cause separate issues for the image alignment and reconstruction routines.

The image series in fig. 5.7 was reconstructed using the EWR software developed as part of this thesis (see chapter 4). The inclusion of a MI based image alignment routine (section 4.3.1.2) as well as the standard cross correlation type

image alignment can be beneficial for graphene like materials as the MI alignment tends to find the correct image alignment in the majority of cases whereas the PCPCF method is prone to generating erroneous alignments in the presence of either mobile polymer residue or completely clean regions with no reference point. The MI method has been used for the drift correction for the image series shown in fig. 5.7.

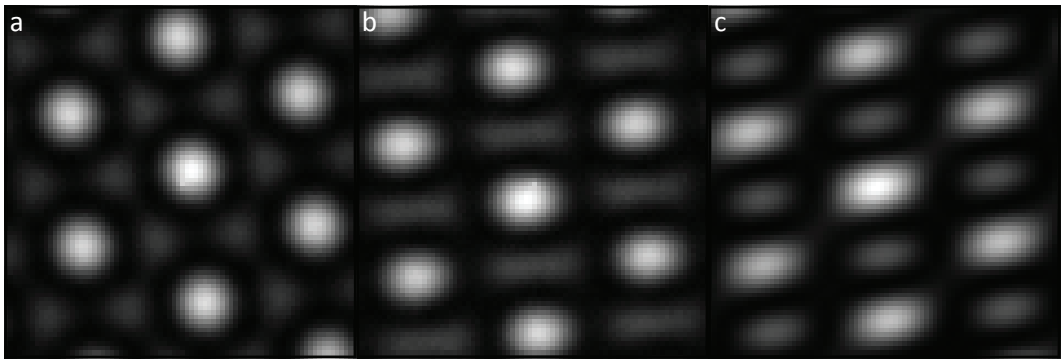

Figure 5.8: Mutual Information maps between consecutive image pairs for the first 4 images from the focal series fig. 5.7. The mutual information is shown for a small range of image displacements centred around zero. The MI map reflects the translation symmetry of the fluorinated graphene structure, however the correct displacement (central spot) is more strongly correlated than the other spots which is important for the automated image alignment. It can clearly be seen that the MI varies strongly even over single pixel displacements.

The mutual information correlation maps shown in fig. 5.8 give a measure of how well the 2 images are aligned to each other at each of the trial displacements. The symmetry within the correlation maps shows that the images can be aligned in several different positions due the translation symmetry of the structure being investigated, however in all 3 cases the central spot has a higher mutual information than the surrounding spots indicating it is the correct image displacement. The size of the image displacement for a subsection of the image series fig. 5.7 is shown in fig. 5.9, it can be seen that the drift is roughly linear in both the x and y directions.

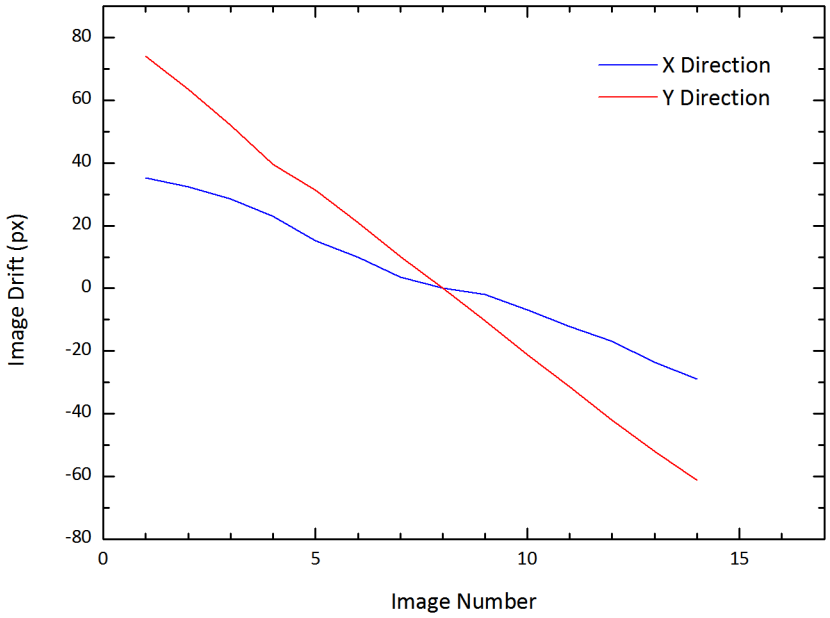

Figure 5.9: Drift measurements for the image series shown in fig. 5.7 showing the measured image displacements in the x and y direction from the selected reference image. These drift values were measured using the MI based image alignment routine, the drift is approximately linear in both directions.

After the image series has been suitably drift corrected the reconstruction process used to generate the exit wave only relies on further information about change in focus level between each image in the series, the absolute focus level of the reference image and the values of the aberrations of the imaging system. In this case the aberration values were recorded using the CEOS aberration correction system software prior to acquisition of the focal series. The absolute focus was initially estimated from assuming the lowest contrast image in the series to be at approximately zero focus, and then it is subsequently refined during the final step of the EWR routine which also detects any residual aberrations and corrects for them.

The resultant exit wave from the series of images shown in fig. 5.7 is displayed in fig. 5.10. The resolution in the exit-wave is sufficient to resolve the hexagonal lattice in much more detail than any of the individual images that make up the focal series and with greatly improved contrast. It is also apparent from the reconstructed phase image that there is significantly increased contrast at strictly alternating carbon positions throughout the entire reconstructed field of view. These positions of increased phase shift are in agreement with the fluorinated graphene being chair C_2F conformation and the increased contrast in the phase image being due to the

presence of fluorine atoms on alternating carbon positions. The modulus image from the reconstruction and an image from the focal series taken at approximately the Scherzer defocus have also been included in fig. 5.10, it is observed that the phase image provides much greater detail than the modulus image from which the presence of increased contrast at alternating positions cannot be inferred. It can also be seen that the reconstructed phase image is of much greater contrast than the scherzer defocus image.

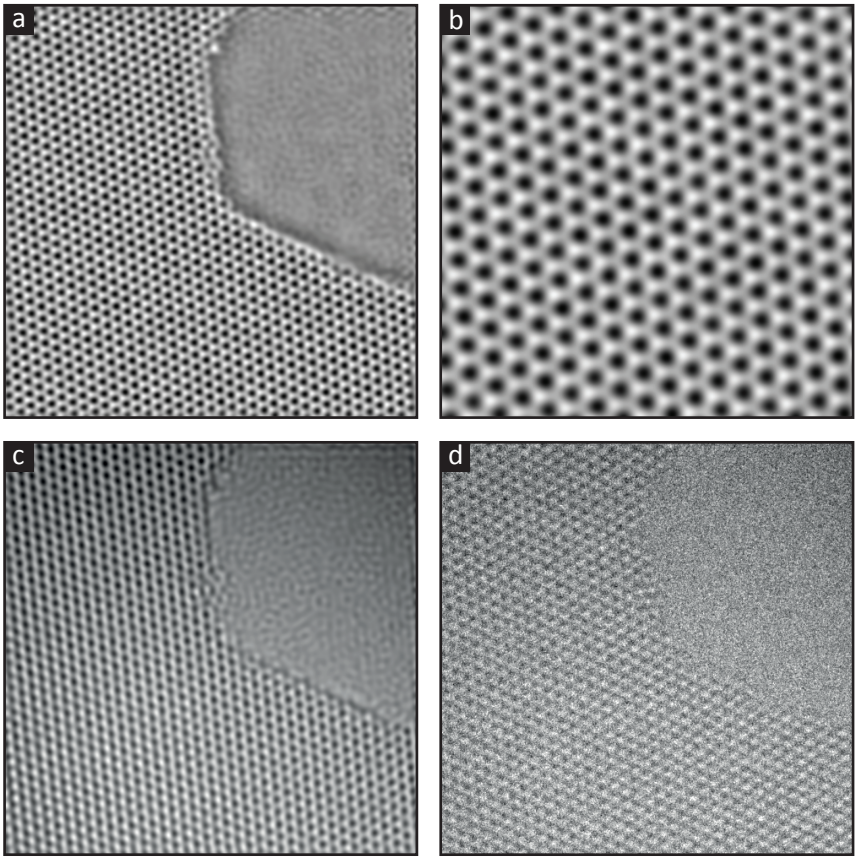

Figure 5.10: a) Exit wave phase reconstructed from the series in fig. 5.7, b) enlarged sub-region from a), c) Exit wave modulus reconstructed from the series in fig. 5.7 and d) an image from the focal series in fig. 5.7 taken at approximately the Scherzer defocus.

5.4.3 Comparison with Simulated Exit Waves

Atomic models of the various fluorinated graphene structures as described in [Sahin et al. 2011] were generated and the multislice software described in chapter 3 was used to simulate focal series for each model under the same approximate imaging conditions as those determined for the acquisition of the experimental focal series

images in fig. 5.7. These simulated focal series were then used as inputs for the exit wave reconstruction procedure to simulate the expected phase image for each of the different structures described in section 5.2. These exit waves can subsequently be compared to the phase images derived from the experimental series to see if they match. Using an exit wave reconstructed from a series of simulated images rather than the simulated exit wave produced by the multislice simulation directly provides a more meaningful comparison as this takes into account the information that should be lost or corrupted by the detection procedure and also inaccuracies in the reconstruction such as those caused by imperfect alignment etc. Small random image displacements were intentionally added to the reconstruction of the simulated series to simulate the effects of the alignment procedure not producing a perfect registration as it is probable that the experimental series will never be perfectly aligned by the reconstruction process unlike the simulated series which is perfectly aligned due to the nature of the simulation process.

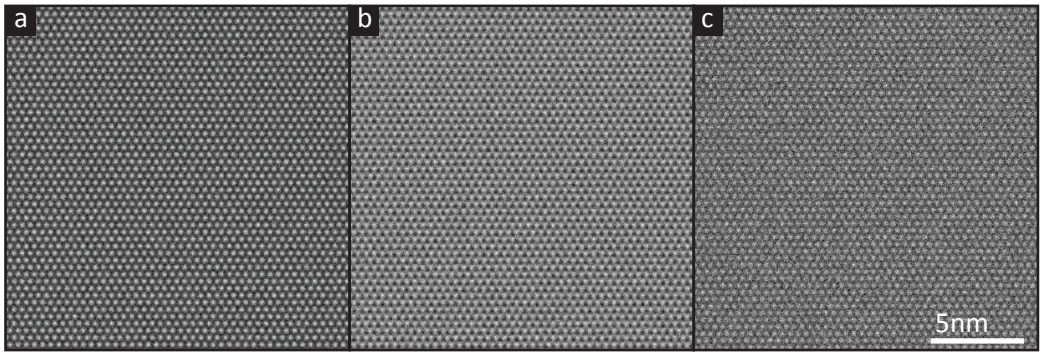

Figure 5.11: Simulated images of a) graphene, b) chair conformation C_2F, and c) stoichiometric fluorographene CF, for an aberration corrected microscope at 80kV with a Spherical aberration of $1\mu m$, defocus $-2nm$ and all other aberrations set to zero. It is hard to distinguish between the image contrast for the three cases from a standard HRTEM image.

The simulated images for boat phase C_2F are not included here as the clear 6-fold symmetry within the images as evidenced by the diffraction pattern in fig. 5.4 clearly precludes the presence of the boat phase. Simulated HRTEM images for graphene, chair C_2F, and fully fluorinated CF show very little variation under standard imaging conditions as shown in fig. 5.11, the presence of fluorination can not be reliably identified from a standard HRTEM image alone for this reason as similar contrast can be achieved by other means.

The reconstructed phase images from these simulated series given in fig. 5.12 do show significant variations for the different fluorination cases, both graphene and CF are qualitatively very similar but there is a much greater phase shift for CF

due to the presence of the additional fluorine atom at every carbon position, this is again shown in the line profiles through these images in fig. 5.12. The chair C_2F is easily distinguishable from both graphene and CF due to the alternating presence of the fluorine atoms causing an increased phase shift at strictly alternating carbon positions just as seen for the experimental reconstruction in fig. 5.10.

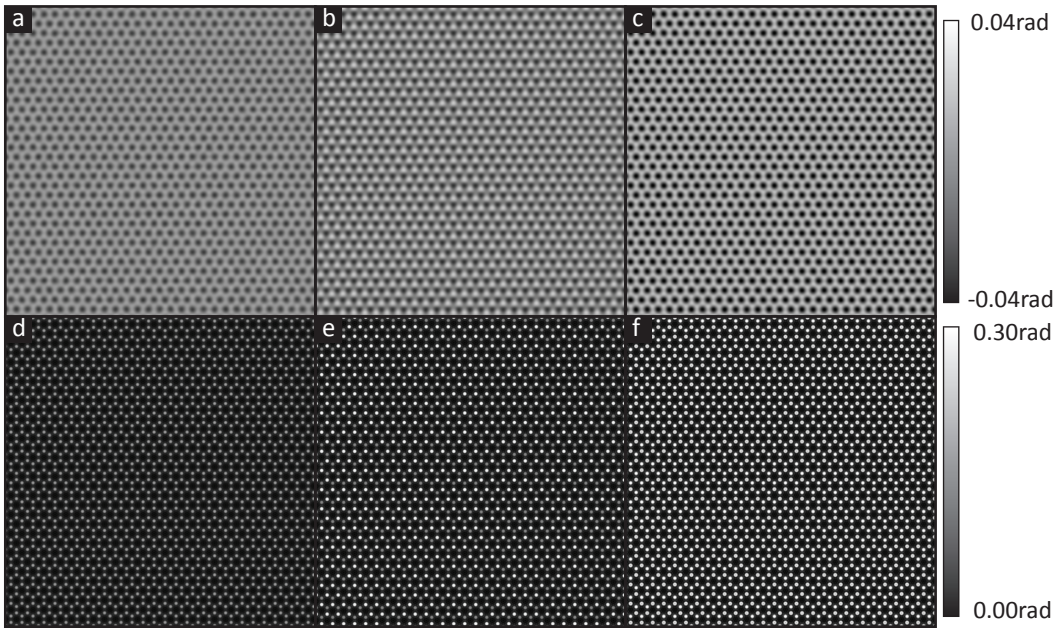

Figure 5.12: Top row: EWR phase images produced from simulated focal series of a) graphene, b) C_2F fluorinated graphene, and c) stoichiometric fluorographene CF. These images are all displayed on the same scale so as to distinguish between the graphene and CF which are otherwise very similar. Bottom row: Phase images taken directly from the multislice simulations used to generate the simulated series for d) graphene, e) C_2F fluorinated graphene, and f) stoichiometric fluorographene CF. The red lines indicate the position of the line profiles shown in fig. 5.13.

The exit waves generated using the multislice procedure for each of the structures are also shown in fig. 5.12, but these are not a useful comparison for performing EWR on a experimental image series as the latter process is affected by noise in the images and the performance of both the EWR procedure and the microscope.

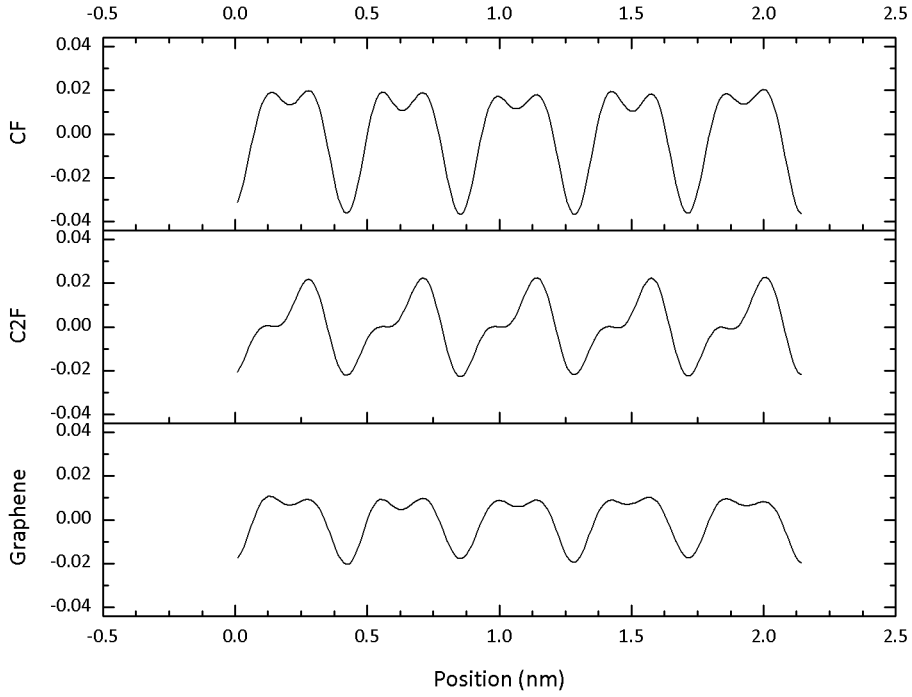

Figure 5.13: Line profiles through the simulated phase images shown in fig. 5.12 for graphene, C_2F fluorinated graphene, and CF fluorinated graphene.

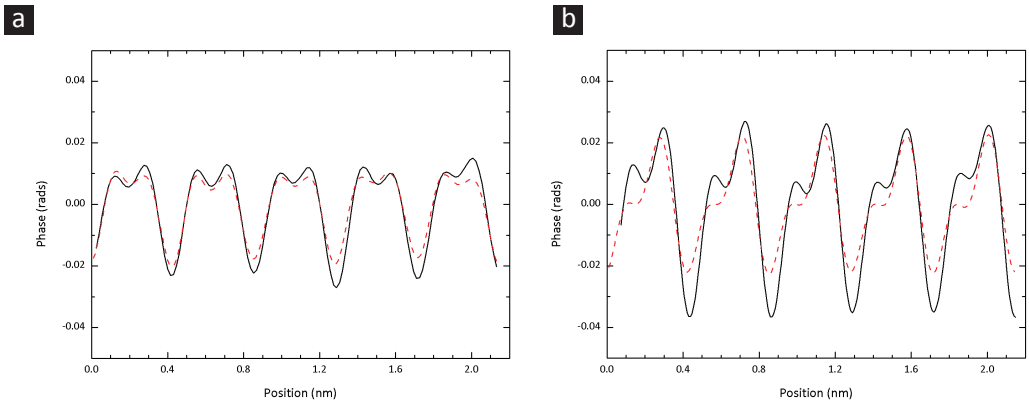

Figure 5.14: Overlaid line profiles from the experimental phase image [black] and the simulated phase image [red] for a) pristine graphene, and b) C_2F fluorinated graphene.

Line profiles through the phase images also show the difference in contrast produced when fluorine is present, a phase shift of approximately $0.025rad$ is found for carbon only in the graphene phase and also in the C_2F, whereas a value of around $0.05rad$ is found for a fluorine and carbon pair in both the C_2F and CF images.

These values are close to those found in the experimental reconstruction as shown in fig. 5.14 where the match between the simulated and experimental results for graphene are exceptionally close and the clear difference between alternating atom sites is shown for the fluorinated graphene as well as the overall increase in phase shift between graphene and its fully fluorinated form. The size of the phase shifts are not in perfect agreement but these values are influenced by the quality of the image alignment and reconstruction process which is inherently difficult to match between experiment and simulation. It is also possible that some of the approximations in the standard multislice simulation routine such as neglecting interactions between atoms are not accurate enough for samples such as fluorinated graphene and thus the exit wave used to generate the simulations could be inaccurate. An exit wave was also simulated using the DFT calculated potentials for the C_2F fluorinated graphene structure as described in section 3.2.2 however fig. 5.15 shows that this approach produced an exit wave with approximately the same general form and relative values as those produced directly via the multislice simulation method. The shapes of the peaks in phase at each atom position in the DFT result do not closely match the characteristic shape of those produced via the multislice method however the DFT result is also at a significantly lower resolution and this level of detail is beyond what could be currently be observed in a reconstructed exit wave.

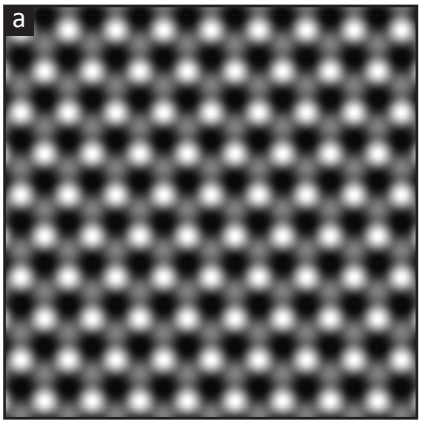

Figure 5.15: Exit wave for C_2F fluorinated graphene produced through application of the multislice procedure to potentials calculated using DFT as covered in section 3.2.2.

5.5 Chapter Summary

EWR of focal series from a fluorinated graphene sample in tandem with image simulations of the various proposed structures have been used to identify the conformation of the fluorinated graphene sample from the phase of the reconstructed

exit wave. EWR produces clear contrast differences at alternating carbon positions which coincide with the image contrast predicted from multislice simulations of a chair conformation C_2F structure. The phase images also suggest the chair (C_2F) is ordered over the entire region within the image field of view.

Chapter 6

Study of Macromolecular Block Copolymer Assemblies via Exit Wave Reconstruction

6.1 Introduction

TEM is a commonly used technique for determining the structures of macromolecular samples, however the weak electron scattering cross-section of carbon generally leads to low contrast images unless they are acquired under a significant defocus, or alternative techniques such as staining are employed [Harris and Scheffler 2002]. Staining involves the use of compounds such as uranyl acetate which are significantly more opaque to electrons and can bind selectively to parts of the sample offering significantly improved contrast. EWR is an alternative approach which allows for high contrast and high resolution imaging of macromolecular samples by combining the data taken from a series of images to produce a phase image, this phase image offers much higher contrast and improved resolution over heavily defocused imaging when used in conjunction with ultra low contrast supports and does not require any additional staining.

6.2 Block Copolymer Assemblies

Amphiphilic block copolymers are polymers which are made up of several polymer blocks which have both hydrophobic and hydrophilic properties. These block copolymers tend to be capable of self-assembling in solution to form a range of different 3D morphologies [Zhang and Eisenberg 1995]. Some of these structures have attracted much attention for their potential uses in catalysis or as drug delivery systems where their ability to encapsulate other species can be advantageous. Control over the

conditions during the assembly process can be used to preferentially drive formation of spherical micelles, cylindrical micelles, polymersomes and other additional structures [Zhang and Eisenberg 1995]. These structures tend to self assemble in solution in an attempt to minimize the interactions between the hydrophobic parts of the macromolecules and the solution itself. It is important to be able to accurately characterise these different morphologies as there is a strong relationship between structure of the assembly and its functional properties.

Some of the different possible morphologies are depicted in fig. 6.1, the micelles consist of a single membrane structure and the polymersomes are enclosed membrane structures with hollowed central regions.

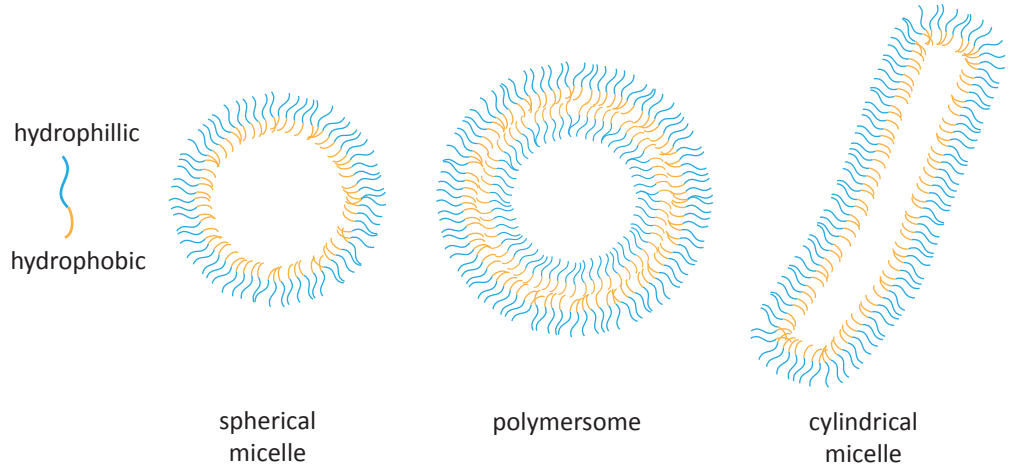

Figure 6.1: Some of the 3D morphologies capable of self assembly from amphiphilic block copolymers.

6.2.1 Conventional Imaging Methods

Common approaches to imaging these self-assembled structures are usually based on light, x-ray and neutron scattering or electron and atomic force microscopy. The advantages of the scattering based techniques are that they can be performed in the solution state, but the results from these techniques are indirect and usually require the fitting of data to a model, these kinds of results are also based around a bulk or average over numerous specimens and not individual macromolecules. In contrast, the electron microscopy and atomic force microscopy techniques make observations on single particles but these techniques have their own inherent difficulties, the particles cannot be imaged in solution and typically have to be dried and placed on a supporting material to be observed. SEM can be used to identify the particles but offers limited resolution in comparison to TEM (an order of magnitude or more),

AFM is able to accurately resolve the 3D structure but it is limited to imaging the particles surface, TEM is capable of the highest spatial resolution but is limited by poor contrast as a result of the weak scattering light elements these particles are typically composed of [Yang et al. 2010; PresaSoto et al. 2010].

This contrast problem is further compounded by the fact that in order to observe the structures in the microscope, the structures must also be supported on some other material. Typical amorphous carbon specimen support films have similar composition and thickness to that of the polymersomes and micelles themselves and consequently produce similar levels of contrast. The traditional solution to this problem is to use some form of staining technique to adsorb materials with a much greater atomic number to the sample or specimen support so that it becomes a stronger scatterer of electrons and is therefore much more visible in the microscope, this staining can however have undesired consequences and generate undesirable artefacts in the images but is nonetheless often necessary to generate sufficient image contrast [Talmon 1983].

Another solution to the low contrast problem is to determine the phase of the electrons that pass though the sample rather than to take a conventional TEM image, in principle this can be achieved in a number of different ways, EWR can be used to combine information from standard images to determine the phase, holography can be used to determine the phase via interference with a reference wave, or a phase plate can be employed to convert the phase differences into visible amplitude contrast. Phase plates have already been used to image organic / biological samples via TEM but the use of phase plates requires additional hardware that may not be present on an electron microscope [Chang et al. 2010; Gamm et al. 2010; Murata et al. 2010; Hall et al. 2011].

6.2.2 Graphene Oxide Specimen Supports

An alternative method to staining that can be used to generate sufficient image contrast in TEM is to drastically lower the contrast from the specimen support instead of attempting to increase the contrast from the particles themselves. Graphene and graphene oxide (GO) have both been shown to be exceptional specimen supports for electron microscopy as shown in fig. 6.2 [Pantelic et al. 2011; 2010]. Graphene is almost entirely electron transparent at low resolutions with just a monolayer of carbon atoms to scatter the electrons, graphene also offers rather impressive mechanical properties that make it an excellent support material. GO retains many of the benefits of graphene such as the excellent electron transparency but can be more convenient to work with as GO specimen support grids are much easier to produce than graphene alternatives. The hydrophilic nature of GO may also be helpful with the adhesion of biological molecules. Whilst GO is also of similar

composition to the micelles and polymersomes, its thickness ensures that it produces far less contrast than the objects it is used to support provided there is a small number of layers.

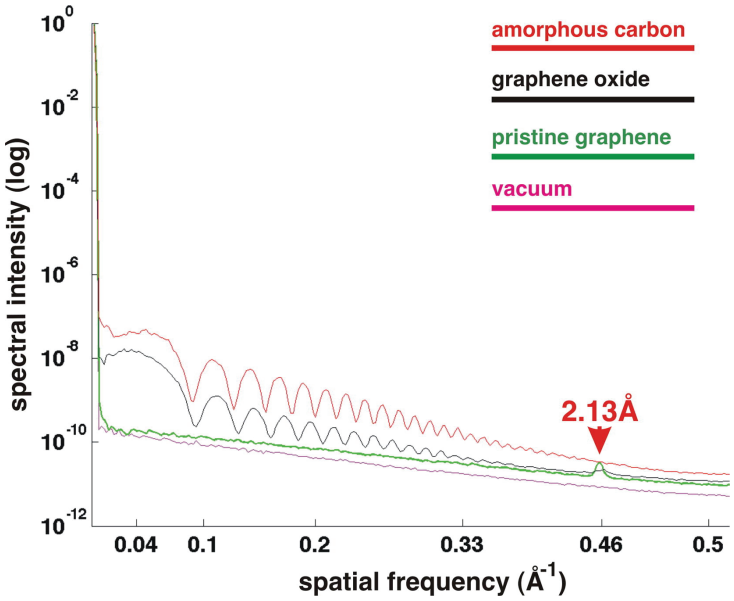

Figure 6.2: Power spectral densities taken from images of different specimen supports including graphene, graphene oxide, thin amorphous carbon and vacuum as a comparison. Taken from [Pantelic et al. 2011]

The use of GO specimen supports for biological molecules can in many cases allow them to be imaged in TEM without any staining procedure applied [Patterson et al. 2012]. The contrast from thin layers of GO is weak enough that the molecules are readily visible in a standard HRTEM image even under relatively low defocus conditions. GO specimen supports should also be beneficial for performing EWR in addition to standard imaging, this is because the phase change of the exit wave is proportional to the integral of the specimen potential the electrons have passed through [Kirkland 2010]. For a monolayer of GO this should be substantially less than for a much thicker layer of amorphous carbon so the GO should also produce little contrast in a reconstructed phase image in addition to standard HRTEM images.

6.3 Focal Series Reconstruction of a Polymersome

In order to resolve interpretable high resolution spatial information about a sample it is usually necessary to image the sample around the Scherzer defocus which gives the a the highest point resolution and a large information transfer passband

upto this point (see eq. (1.13)). The Scherzer defocus is usually a relatively small amount of underfocus and therefore may not offer much contrast which can limit the information that can be extracted from an image. Imaging under a greater defocus improves the contrast, but this involves a sacrifice in the available point resolution. Information beyond the point resolution is difficult to interpret due to the constant sign changes in the oscillating transfer function.This is described effectively by the PCTF which details the spatial frequencies at which information will be present within an image based on the imaging conditions under which it is acquired. In an aberrated microscope this function has points at which its value is reduced to zero at which there will be no information transferred to the image and these will generally be more frequent at greater defocus.

EWR takes the information from multiple images at different defocii to reconstruct a single image which has information transfer at all spatial frequencies that are present in at least one image within the focal series. This allows it to achieve a higher resolution when compared to an in focus image as information is recovered up to the information limit rather than just the point resolution. EWR also recovers the phase of the exit wave, the phase image can be sensitive to the lighter elements which also allows for high contrast imaging if the imaging conditions within the focal series can be characterised accurately enough to perform the reconstruction.

6.3.1 Experimental Focal Series

A focal series of a polymersome of the block copolymer poly(acrylic acid)$_{11}$-b-poly(styrene)$_{250}$ on a GO specimen support is shown in fig. 6.3. This series consists of 40 images taken with a focal step of $25nm$ between images. The images were acquired on an aberration corrected JEOL ARM-200F at $80kV$ with a spherical aberration tuned to $\sim 1\mu m$. The full table of aberration coefficients taken from the CEOS aberration corrector prior to series acquisition are given in table 6.1. The defocus step was calibrated for the microscope prior to the series acquisition using the diffractogram matching routine explained in section 4.4.

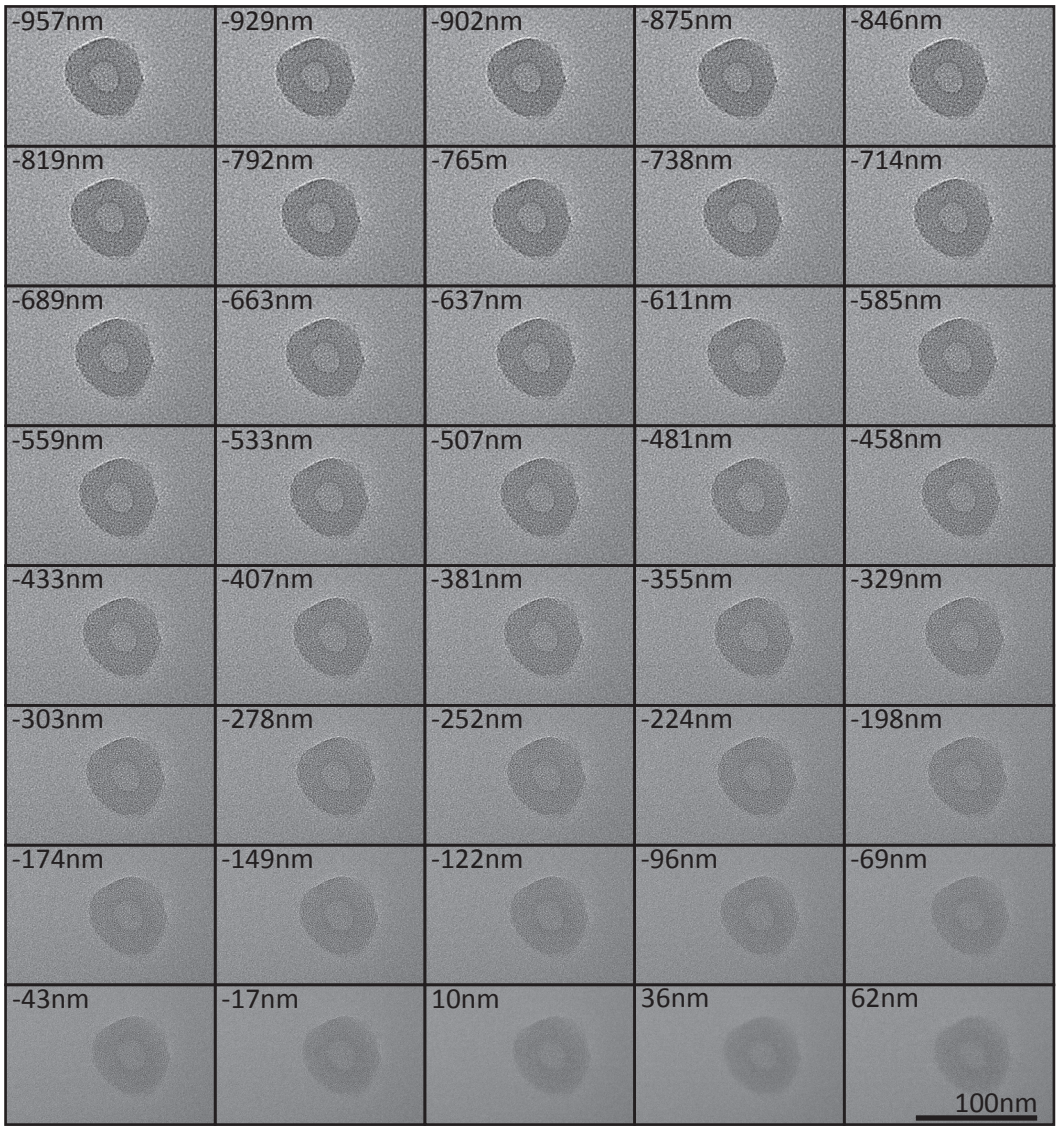

Figure 6.3: A focal series of images of a poly(acrylic acid)$_{11}$-b-poly(styrene)$_{250}$ polymersome used for exit wave reconstruction. This series of 40 images were acquired with a nominal focus increment of $25nm$ (focus values are indicated in the corner of each image). The series is shown in row-major order (the most under focused image is top left). There is a substantial drift (left to right) over the course of the series.

It is clear to see from the series in fig. 6.3 (which are all displayed on the same contrast scale) that the underfocused images have much stronger contrast but lack the fine detail that is present in the images closer to focus. Figure 6.4 shows the power spectral density (PSD) taken from a selection of the images within the series along side the PCTF calculated for the imaging conditions determined for each image. The PCTF shows the predicted information transfer within each image based

Aberration Coefficient	Value
A_2	$13.4nm$ $(178.5°)$
B_2	$16.96nm$ $(-4.4°)$
A_3	$899.5nm$ $(-81.8°)$
S_3	$754.4nm$ $(-170.2°)$
C_3	$606nm$
A_4	$38.1\mu m$ $(-8.6°)$

Table 6.1: Table of aberration coefficients recorded prior the acquisition of the image series in fig. 6.3 using the CEOS aberration corrector software.

upon the image conditions determined using the EWR algorithm and the PSD shows the distribution of actual information contained within the image determined from the intensities of spatial frequencies in a Fourier transform of the image. The point resolution of each image (measured by the position of the first minima) increases as the defocus moves closer to zero, this means that information up to this frequency is all transferred to the image with the same sign, the sign then oscillates between the subsequent minima which makes the highest resolution information in the images difficult to interpret. The excellent agreement between the predicted PCTF and normalised PSD shows that the defocus determination routines within the EWR procedure are able to accurately characterise these images which is necessary for an accurate reconstruction.

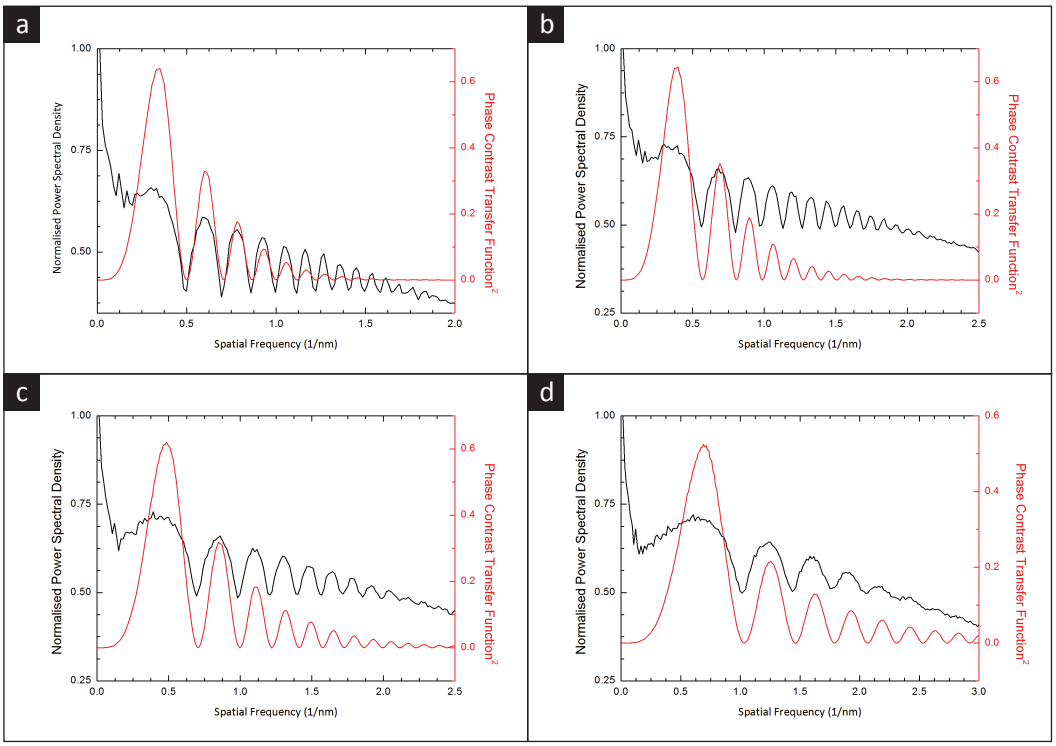

Figure 6.4: Predicted phase contrast transfer functions for several images from the series in fig. 6.3 plotted against the measured power spectral density from the image Fourier transforms. The phase contrast transfer functions were generated based on the imaging conditions determined during the exit wave reconstruction without including the effects of astigmatism: a) -957nm, b) -738nm, c) -481nm and d) -224nm.

6.3.2 Phase Restoration

The exit wave reconstruction produced from the image series in fig. 6.3 is given in fig. 6.7. The drifts calculated for the image series alignment are shown in fig. 6.5 plotted in the order in which the images were acquired. A total drift of over 500 pixels in the x-direction and less than 50 pixels in the y-direction is measured and accounted for across the entire series. In this case the image drift is non linear for the first few images in the series before settling down into a linear drift after a short while.

The phase image here shows very strong contrast between the polymersome and the underlying GO support when compared to any of the individual HRTEM images in the focal series allowing for a much easier determination of the size of features within the image. The enlarged region in fig. 6.7 also shows the presence of high resolution information resolving the corona region of the polymersome, this area is very hard to distinguish in any of the individual HRTEM images as evidenced by the enlarged region of the same area from the in focus image from the original

focal series. The interface between the hollow central region of the polymersome and the outer region is also much easier to distinguish in the phase image than in the near to focus image. When comparing the FFTs of the reconstructed image with the images in the original focal series (fig. 6.6), the reconstructed image has a much more uniform information transfer out to higher spatial frequencies than any of the individual images. The reconstructed image also shows no missing information bands which is a characteristic of all of the defocused images within the original series.

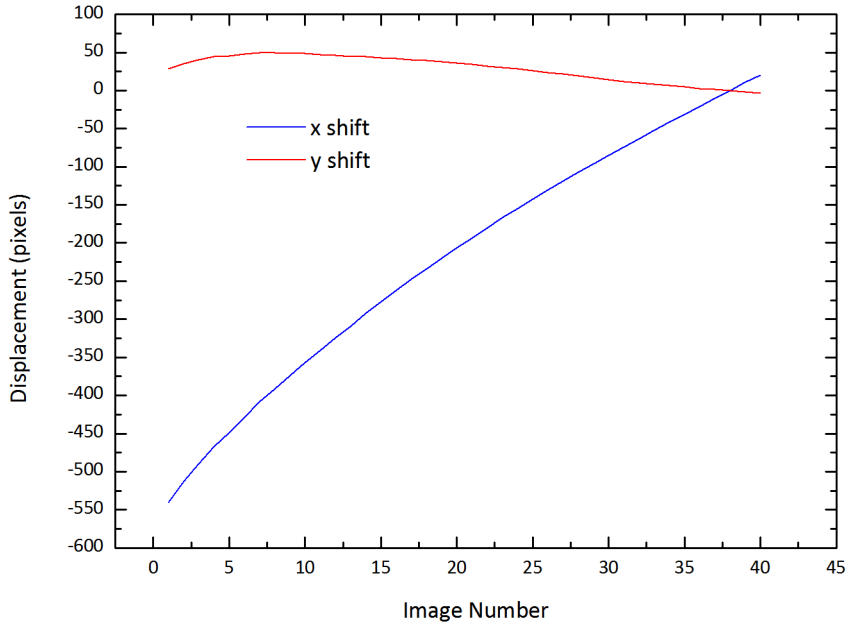

Figure 6.5: Image displacement measurements calculated for the image series fig. 6.3 by the EWR plugin.

Whilst the improvement in resolution and removal of missing information bands that result from EWR are both beneficial to improving the images of the polymersome, the main advantage in performing EWR is gaining access to the phase of the electron wave. Because thin samples made of weak electron scattering materials primarily alter the phase of the electrons rather than the amplitude, the phase image is much more sensitive to the differences between the sample and specimen support than the amplitude and as such there is far greater contrast in the phase of the exit wave than there is in the amplitude of the exit wave. The traditional approaches to exploit this would be to image under a large defocus to convert the phase difference to amplitude contrast, or to use a physical phase plate or electron holography [Danev and Nagayama 2001; Nagayama and Danev 2008; Lichte 1986]. The former approach

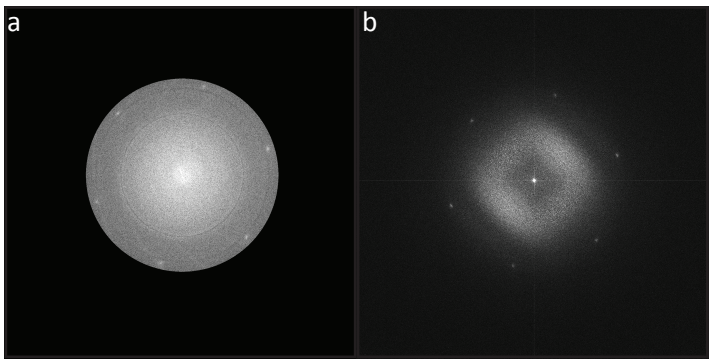

Figure 6.6: a) FFT of the reconstructed image from the image series fig. 6.3, b) FFT from image 37 in the series fig. 6.3. An intentional cutoff has been applied to the reconstruction at $5nm^{-1}$ just past the position of the first order graphene lattice reflections to minimise the inclusion of noisy information at higher spatial frequencies where the MTF of the CCD is much lower.

greatly limits the resolution as explained earlier, and the other approaches require potential modifications to microscope hardware and so are significantly more complex to undertake than collecting a focal series which is a simple procedure requiring no hardware modifications.

6.3.3 Verification of Results

It is worth noting that the acquisition of a focal series containing 40 images is likely to have undesirable consequences on the sample area being investigated due to the prolonged exposure time required to take so many images, particularly for biological samples that are more susceptible to beam damage. These images were acquired at a relatively low 80kV to minimise knock-on damage from the electron beam but there is still likely to be some damage caused, especially for biological samples which are typically required to be imaged under low dose or cryo conditions for this reason.

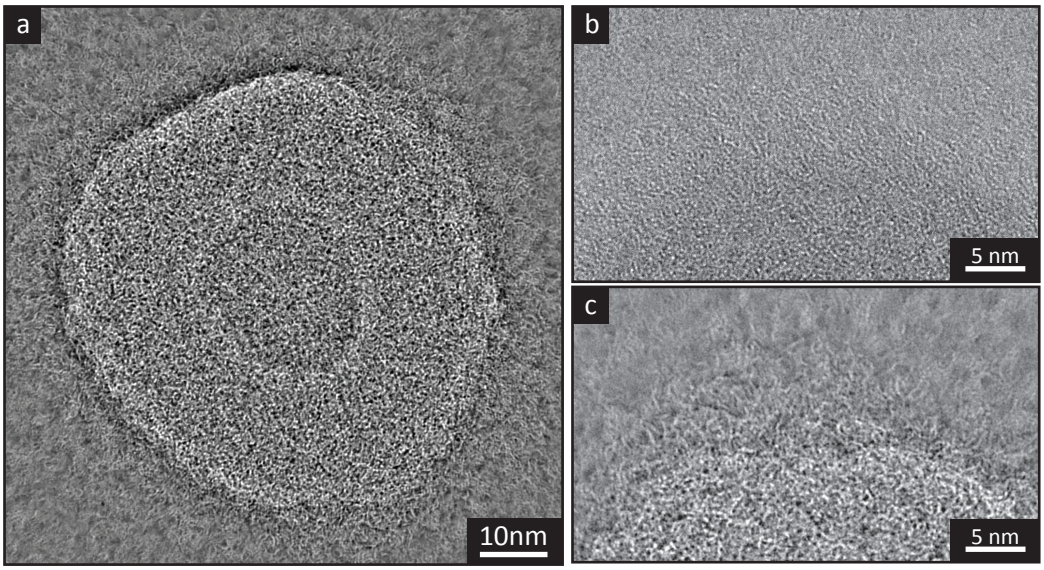

Figure 6.7: a) Phase image from EWR of image series in fig. 6.3, b) enlarged region of in focus image from fig. 6.3, and c) enlargement of same region as in b) from the EWR phase image.

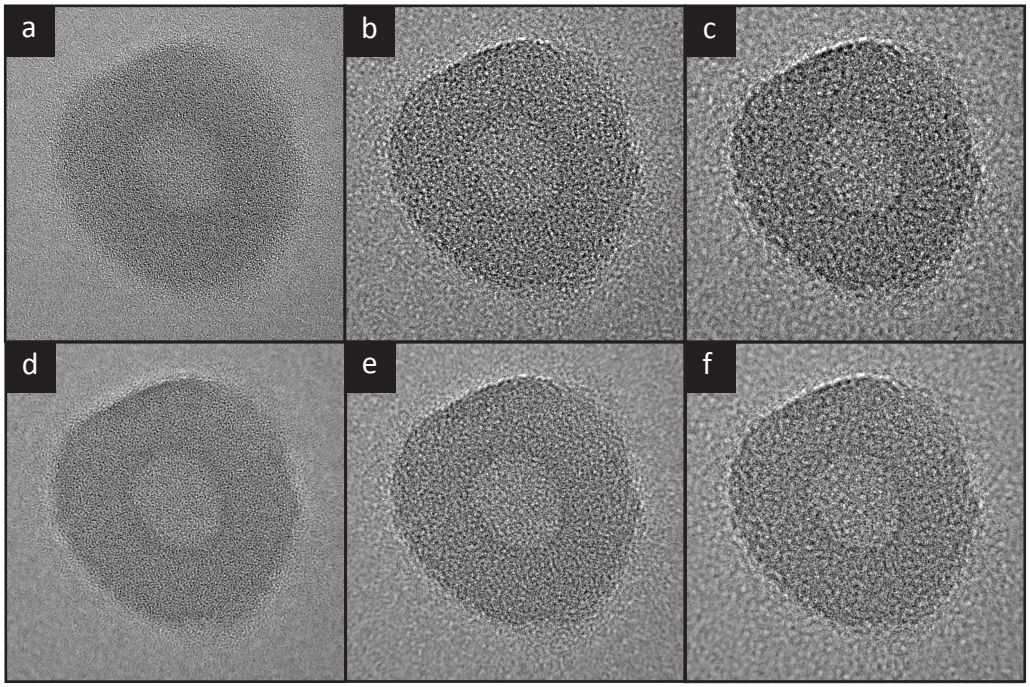

Figure 6.8: a)-c) Original images from the focal series in fig. 6.3 at +62nm, -202nm and -436nm respectively. d)-f) Predicted images generated from the exit wave reconstruction in fig. 6.7 for the same conditions determined for a)-c) respectively.

It is important to have some way to determine if the exit wave generated from a series of images is a reliable representation of the sample, this can be achieved in a number of ways. Firstly, it is possible to reproduce a predicted image for any specified imaging conditions derived from the information contained within a reconstructed exit wave. The original focal series in its entirety can be simulated using the reconstructed exit wave and knowledge of the focus level of each image. These simulated images can then subsequently be compared to the images from the original series. If there are significant deviations between the simulated images and the original images then this suggest that the exit wave does not reliably describe the sample in question or the approximations inherent in the procedure are not justified for this sample. If the simulated images match with the original images then it can be assumed the reconstruction process has been successful. This process has been carried out for some of the images within the polymersome focal series in fig. 6.3 and the results are shown in fig. 6.8 where it can be seen the simulated images match the originals with a high degree of fidelity. Secondly, as it is possible to reconstruct an exit wave from as little as 3 images using the FTSR method, a long focal series may be split into two or more distinct series (odd numbered and even numbered etc.) and a reconstruction carried out for each sub-series. These independent reconstructions can then be compared against each other and should have a high degree of similarity as seen for the two reconstruction in fig. 6.9 where alternating images have been separated to form two different series where the difference between the two reconstructions is very small.

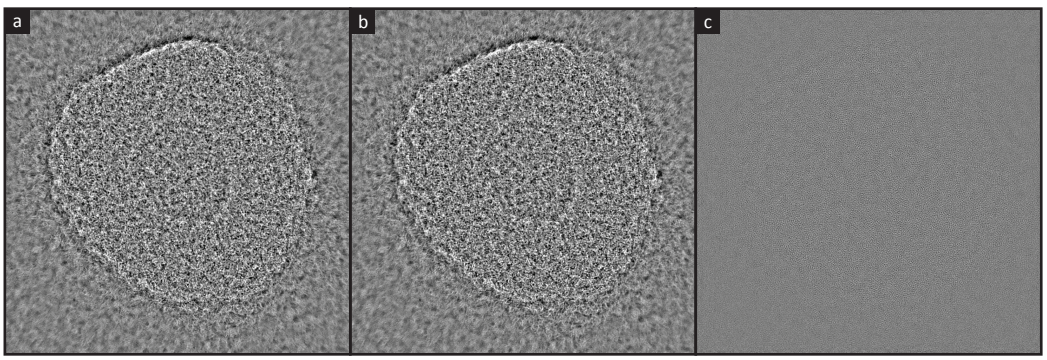

Figure 6.9: a) Phase image from a focal series reconstruction of the even numbered images in fig. 6.3, b) Phase image from a focal series reconstruction of the odd numbered images in fig. 6.3, c) The difference between the two phase images a) and b) plotted on the same scale as a) and b).

6.4 Focal Series Reconstruction of Cylindrical Micelle

Two image series of cylindrical micelles formed from the block copolymer Poly(acrylic-acid)$_{333}$-b-Poly(L-lactide)$_{37}$ are shown in fig. 6.10 and fig. 6.11. These series of images have been acquired at 80kV on an aberration corrected JEOL ARM-200F operated with a spherical aberration tuned to $\sim 1\mu m$. The full set of aberration parameters determined prior to acquisition are recorded in table 6.2 and table 6.3. In fig. 6.10 the micelle is supported on monolayer GO as described earlier, and in fig. 6.11 the micelle is supported on a traditional thin amorphous carbon support to illustrate the difference between the two specimen supports.

Aberration Coefficient	Value
A_2	$44.4nm$ $(-151.4°)$
B_2	$11.16nm$ $(-104.5°)$
A_3	$549.6nm$ $(141.8°)$
S_3	$337.3nm$ $(26.8°)$
C_3	$380.2nm$
A_4	$33.73\mu m$ $(153.7°)$

Table 6.2: Table of aberration coefficients recorded prior the acquisition of the image series in fig. 6.10 using the CEOS aberration corrector software.

Aberration Coefficient	Value
A_2	$45.06nm$ $(59.3°)$
B_2	$17.83nm$ $(-135.7°)$
A_3	$745.9nm$ $(113.8°)$
S_3	$764.4nm$ $(42.4°)$
C_3	$461.7nm$
A_4	$20.72\mu m$ $(161.3°)$

Table 6.3: Table of aberration coefficients recorded prior the acquisition of the image series in fig. 6.11 using the CEOS aberration corrector software.

6.4.1 Experimental Focal Series on Graphene Oxide Support

The cylindrical micelle supported on GO (fig. 6.10) can clearly be resolved without the need for any staining even under low defocus conditions. As the defocus is increased the micelle becomes much more visible but some of the fine detail is lost from the image as expected. It is clear from the images in fig. 6.10 that some of the internal structure of the micelle is being revealed in the image but the image contrast is very hard to interpret directly due to the oscillations within the contrast transfer function between the point resolution and information limit.

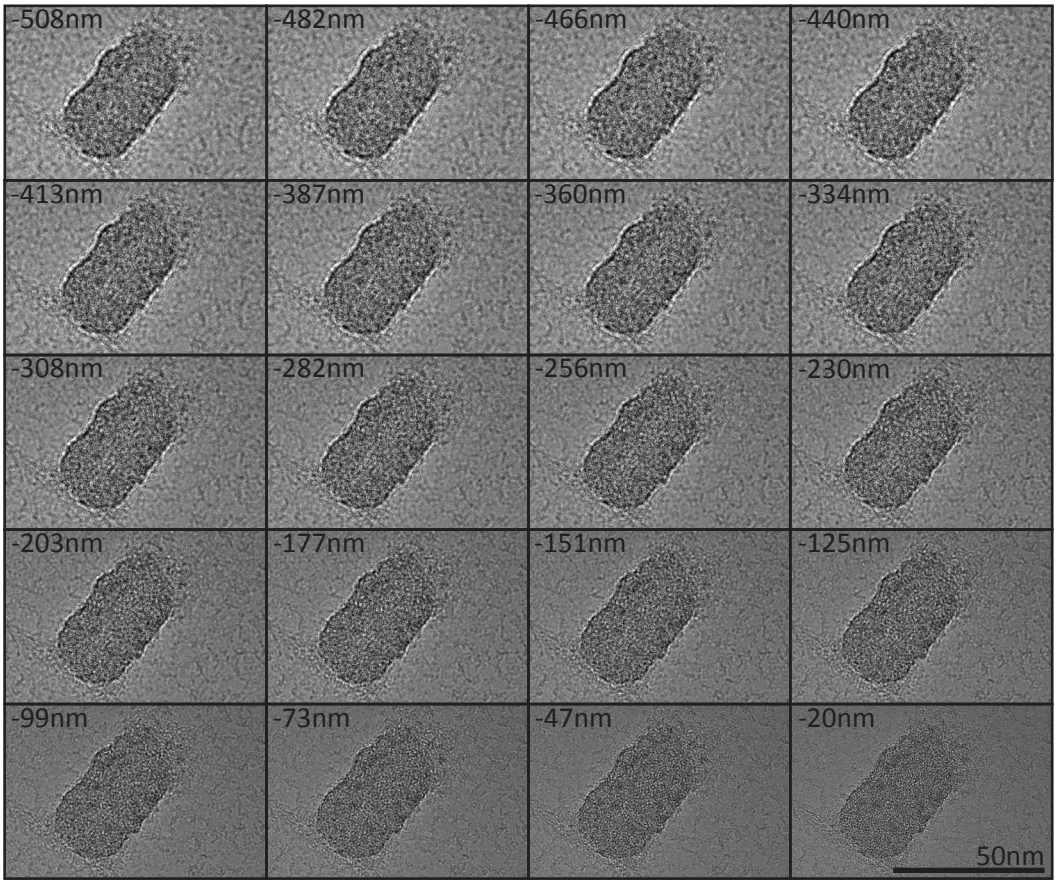

Figure 6.10: A focal series of images of a cylindrical Poly(acrylic-acid)$_{333}$-b-Poly(L-lactide)$_{37}$ micelle used for exit wave reconstruction. This series of 20 images were taken with a nominal focus increment of 25nm (focus values are indicated in the corner of each image). The series is displayed in row-major order (the most under focused image is top left).

6.4.2 Experimental Focal Series on Amorphous Carbon Support

Whilst the position of the cylindrical micelle on the amorphous carbon support (fig. 6.11) is easy enough to see under a large defocus it is difficult to locate the edges of the micelle with any accuracy as there is much less contrast differentiation between micelle and support than in the images of a similar micelle supported on GO. The micelle is only visible as a darker region within the image and very little information about the internal structure is visible within the image due to the strong contrast features also coming from the underlying amorphous carbon.

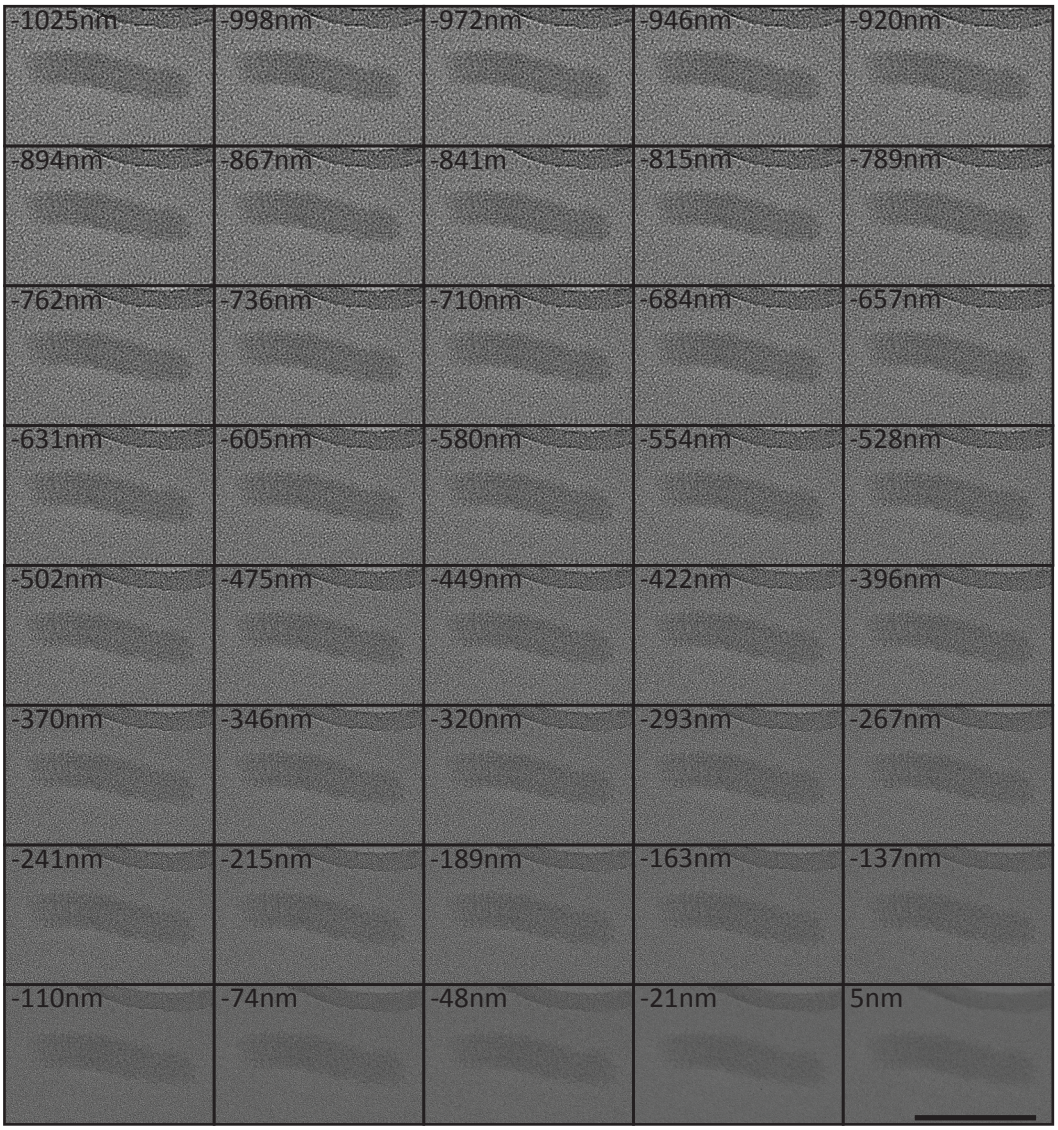

Figure 6.11: A focal series of images of a cylindrical Poly(acrylic-acid)$_{333}$-b-Poly(L-lactide)$_{37}$ micelle used for exit wave reconstruction. This series of 40 images were taken with a nominal focus increment of 25nm (focus values are indicated in the corner of each image). The series is displayed in row-major order (the most under focused image is top left).

6.4.3 Phase Restorations of Cylindrical Micelles

Figure 6.12 shows the reconstructed phase images generated from EWR of both of the focal series of cylindrical micelles on GO and amorphous carbon respectively. The phase image on GO shows a clearly resolved micelle structure with very strong contrast and we are able to resolve internal features within the micelle and a detailed edge structure showing two confined sides and two 'open' ends that could not

reasonably be determined from the HRTEM images alone. This reconstruction shows a substantial contrast improvement from the original focal series images and the phase image is far easier to interpret due to the lack of contrast transfer oscillations. The reconstruction on the thin amorphous carbon support however shows almost no contrast at all between the thin amorphous carbon support and the cylindrical micelle, it is consequently very difficult to even locate the micelle which runs horizontally through the centre of the phase image and accurately determine its boundaries, much less any internal information about its structure. This result shows that the ultra low contrast GO support grids are necessary to perform EWR on low contrast specimens like these self assembled block copolymers where there would otherwise be little difference between specimen and support. Specimen supports such as GO are also beneficial in other ways for EWR as the amorphous content can be used for aberration corrector alignment procedures, and the hexagonal diffraction spots can also be used as an inbuilt size calibration.

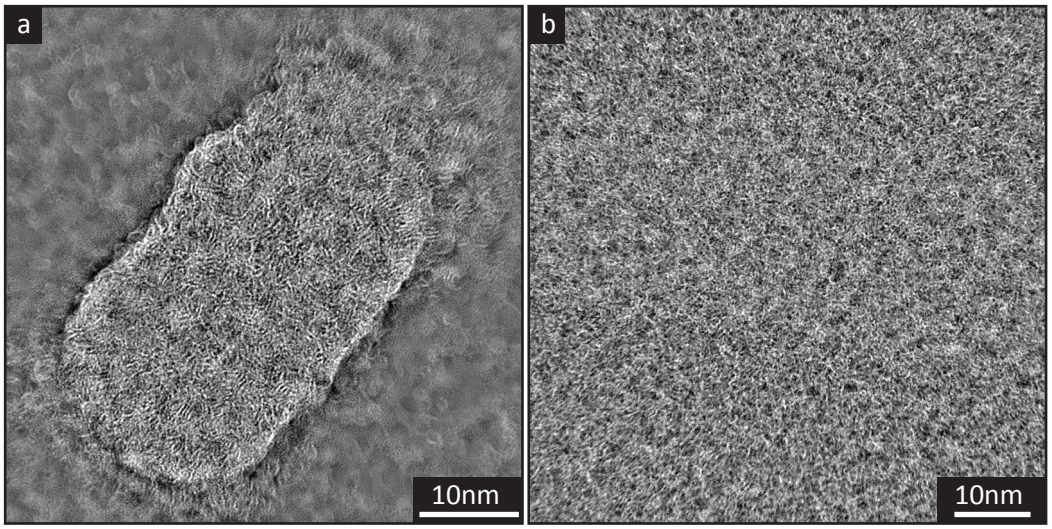

Figure 6.12: a) Phase image from EWR of cylindrical micelle on GO from image series in fig. 6.10, and b) Phase image from EWR of cylindrical micelle on thin amorphous carbon from image series in fig. 6.11.

6.5 Chapter Summary

EWR in conjunction with ultra low contrast GO supports has been used to image self assembled block copolymer assemblies with high resolution and high contrast in TEM. EWR allows us to generate images of biological systems with an unprecedented level of detail without necessitating the use of techniques such as staining which can have other undesired consequences. These images show the capability to resolve details of the substructure within the molecules, i.e. not just the size and shape,

provided that the damage due to the electron beam is not too great. These results are also practically very simple to achieve utilising the EWR software described in chapter 4 and this technique can be implemented on almost any microscope.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This section summarises the main results from this thesis. In chapter 3 software was developed for producing multislice simulations of TEM images from atomic structure models with an emphasis on increasing the accuracy, speed and including detector characteristics to produce more realistic simulations. In chapter 4 software was developed for performing EWR again with a large emphasis on increasing the speed at which reconstructions are able to be performed. In chapter 5 the developed software was used to produce simulated images and reconstructed images from both real and experimental image series of fluorinated graphene to determine the extent and conformation of fluorination. In chapter 6 the EWR software was used to produce reconstructed phase images of self-assembled block copolymer structures to reveal levels of detail unobtainable through traditional imaging methods.

7.1.1 Multislice Software

- A multislice image simulation software package has been developed which supports simulation of TEM, STEM and CBED images.
- The multislice simulation software is capable of greatly accelerating the speed of the simulation via performing computation on GPUs where available with speed increases of between 20 and 100×.
- The simulations can also be performed taking account of detector response via MTF/NPS and noise via the electron dose.
- The limitation on minimum slice thickness has been removed from the multislice procedure and replaced with a more accurate integration accounting for the z-position of the atoms.

7.1.2 EWR Software

- EWR via the FTSR procedure has been developed as a plugin to Digital Micrograph.
- This plugin includes a customizable image acquisition script which includes the option to adjust the focus via voltage variation, blank the beam, delay between images or acquire images in constant acquisition mode.
- Focal series image alignment can be performed via Phase Correlation or via Mutual Information based methods.
- The EWR procedure is written using OpenCL which allows it to run in parallel on multi-core CPUs or high-performance GPUs for performance improvements of between 20 and 75×.

7.1.3 Experimental Reconstructions

- EWR has been performed on cylindrical micelle and polymersome amphiphilic block copolymer structures revealing details beyond those available through standard TEM imaging.
- EWR has also been performed on graphene and fluorinated graphene samples revealing the extent and conformation of the fluorination through comparison with simulations.

7.2 Future Work

7.2.1 EWR - Real Time Combined Acquisition and Reconstruction

The use of GPU computing allows EWR to be performed in a matter of seconds using sufficiently powerful hardware. By using GPUs for the computation, this kind of hardware can easily be installed inside a standard PC used for microscope operation.

To further increase the speed of the reconstruction, the reconstruction process can begin as soon as the first 2 images have been acquired. Provided that the alignment and addition of images to the reconstruction can be performed in less time than the acquisition of a successive image (which is possible using GPUs), this will generate a reconstructed image almost instantaneously after the acquisition of the final image. In this way the speed of EWR may become limited by the speed at which focal series acquisition can be performed, either by the camera frame rate or response of the microscope lenses to focus changes.

The EWR procedure could also be modified to run in a continuous loop. In this way a live image would be displayed at the same frame rate as the live camera

view showing a reconstruction from a moving window of the past N images. The computational performance of the current EWR plugin is already sufficient for this at lower resolutions, and the live image acquisition is also implemented. If necessary the performance of the EWR procedure could also be improved by adapting it towards computation over multiple GPUs with relatively little difficulty.

7.2.2 Multi-GPU Multislice Simulation

The multislice simulation routine adapted to GPUs outperforms similar simulations performed on traditional hardware by a large margin. In the case of TEM image simulation, this allows for simulations of large structures at high resolutions on reasonable timescales using a typical desktop computer ($1,500,000$ atoms at 4096×4096 pixels in around 5 minutes). The majority of the time spent performing the multislice calculation is during the potential calculations for a given slice, which could all be performed in parallel given a greater number of computation resources. This means the performance of the multislice routine could still be increased further by adapting it to computers with multiple GPU's or possibly heterogeneous computing where some of the simpler tasks are performed on the CPU while the GPU is working on something different. These approached have not yet been explored for this simulation software.

7.2.3 Multislice Optimisation for STEM

In the case of STEM image simulation where the time taken to simulate a single STEM pixel is equivalent to an entire TEM multislice simulation, small performance improvements can still make a huge difference to the time taken to perform a simulation. The current implementation for STEM images is inefficient as the slice potentials are recalculated for every STEM pixel despite the fact that they do not change, this is because the potentials for a large number of slices are too large to be conveniently stored on the GPU (500 slices of 2048×2048 pixels is 16GB of data). An alternative approach would be to store several wavefunctions for different STEM probe positions on the GPU (as many as possible), and propagate a group of wavefunctions through each slice at once so that each potential calculation is reused for a large number of probe positions greatly increasing the efficiency. For smaller calculations where the slice potentials can be saved, this should also be implemented as it will greatly improve the performance.

Appendix A

GPU Kernel Listings

A.1 Multislice Simulation Functions

The following segments of code are functions designed to be run on the GPU taken from the full multislice simulation software which is too large to present here in its entirety. The code use for sorting the atoms, as well as for determining the transmission functions are presented here as they are the most important to the functioning of the program. The full source code listing of the program can be found at https://github.com/ADyson/CUDAMultislice. This project has been superseded by an alternative version of the software which has been ported to OpenCL so as to support a much larger range of devices, this can also be found at https://github.com/ADyson/clTEM, although the kernels provide the same functionality in the alternative programming language.

A.1.1 Atom Sorting Code

```
__global__ void atombin(float* xpos, float* ypos, int length, float maxx, float minx, float maxy,
    float miny, int xBlocks, int yBlocks, int * bids, int * zids, float * zpos, float maxz,
    float dz,int nSlices)
{
   int xIndex = threadIdx.x + blockIdx.x * blockDim.x;

   if (xIndex >= length)
      return ;

   int bidx = floor((xpos[xIndex] - minx)/(maxx-minx)*xBlocks);
   int bidy = floor((ypos[xIndex] - miny)/(maxy-miny)*yBlocks);

   int zid  = floor((maxz-zpos[xIndex])/dz);
   zid-=(zid==nSlices);

   bidx-=(bidx==xBlocks);
   bidy-=(bidy==yBlocks);

   int bid = bidx + xBlocks*bidy;
```

```
20
21      bids[xIndex] = bid;
22      zids[xIndex] = zid;
23  }
```

This GPU kernel divides the atoms into 'bins' of atoms in a similar spatial position which is used to optimise later sections of the software. The group of atoms are divided into an equally spaced grid in the x and y directions and into 1Åsteps in the z-direction. The atoms will then be stored in GPU memory in order of z then y then x bin so that groups of atoms with similar z and y coordinates and increasing x coordinates will all be stored consecutively in memory. Heavy use of this particular ordering system is made in optimising the calculation of the transmission function for the IMS and CMS methods.

A.1.2 Transmission Function Kernel - Finite Difference Method

```
1   __global__ void binnedAtomPKernelFD(cuComplex* V, float pixelscale, const float* devfparams,
2      float* devAtomZPos, float* devAtomXPos, float* devAtomYPos, int* devAtomZNum,
3      int* devBlockStartPositions, int dz,   float z, int nSlices)
4   {
5      // Function which uses parameterisation to determine potential at r from element z
6      float vatom(int Z, float radius, const float * devfparams);
7      float sumz = 0; // To store value of potential, initialized to zero.
8      float rad2 = 0;
9
10     int xIndex = threadIdx.x + blockIdx.x * blockDim.x;
11     int yIndex = threadIdx.y + blockIdx.y * blockDim.y;
12
13     if (xIndex >= res[0] || yIndex >= res[1])
14        return ;
15
16     int Index = (yIndex * res[0]) + xIndex;
17
18     // Determine the ID's of the slices that need to be included
19     int topzid  = floor((maxs[4]-maxs[5]-z)/dz) - consts[2];
20     int bottomzid = floor((maxs[4]-maxs[5]-z)/dz) + consts[2];
21     if (topzid < 0) topzid = 0;
22     if (bottomzid >= nSlices) bottomzid = nSlices-1;
23
24     // consts[] contains number of blocks and slices to loads in constant memory, 0=x,1=y,2=z.
25
26     // The next two lines calculate which atom blockID's to load based on this threads blockID,
27     // complicated conversion is required because the number of blocks in each direction (atoms)
28     // does not correlate with the number of blocks in each direction (threads). Also the
29     // area covered by each block takes into account the fact the blocks include extra threads
30     // which overhang the end of the sample if it does not perfectly divide by the block size.
31     // This should scale correctly if the number of blocks is ever changed :)
32
33     for(int k = topzid; k <= bottomzid; k++)
34     {
35        for (int j = floor(float((blockIdx.y*blocks[1]*bins[1]*pixelscale/ (maxs[2]-maxs[3])))
36           - consts[1] ); j <= ceil(float(((blockIdx.y+1)*blocks[1]*bins[1]*pixelscale /
```

```
37          ( maxs[2] - maxs[3] ))) + consts[1]); j++)
38       {
39          for (int i = floor(float((blockIdx.x*bins[0]*blocks[0]*pixelscale / (maxs[0]-maxs[1])))
40           - consts[0] ); i <= ceil(float(((blockIdx.x+1)*bins[0]*blocks[0]*pixelscale /
41           ( maxs[0] - maxs[1] ))) + consts[0]); i++)
42          {
43             // Check bounds to avoid unnecessarily loading blocks when i am at edge of sample.
44             if(0 <= j && j < bins[1])
45             {
46                // Check bounds to avoid unnecessarily loading blocks when i am at edge of sample.
47                if (0 <= i && i < bins[0] )
48                {
49                   // Check if there is an atom in bin, arrays are not overwritten when there
50                   // are no extra atoms so if you don't check could add contribution more than once.
51                   for (int l = devBlockStartPositions[k*bins[0]*bins[1] + bins[0]*j + i];
52                      l < devBlockStartPositions[k*bins[0]*bins[1] + bins[0]*j + i+1]; l++)
53                   {
54                         rad2 = (xIndex*pixelscale-devAtomXPos[l])*(xIndex*pixelscale-devAtomXPos[l])
55                         + (yIndex*pixelscale-devAtomYPos[l])*(yIndex*pixelscale-devAtomYPos[l])
56                         + (z-devAtomZPos[l])*(z-devAtomZPos[l]);
57
58                         if( rad2 < 5.0f) // Check atom is within specified range.
59                            sumz += vatom(devAtomZNum[l] , rad2 , devfparams);
60
61                   }
62                }
63             }
64          }
65       }
66    }
67    V[Index].x = sumz;
68    V[Index].y = 0;
69 }
```

This GPU kernel calculates the potential at each pixel from all the atoms within a defined cutoff radius. It is used for the finite difference version of multislice and uses the subroutine below for calculating the potential from each individual atom instead of the projected potential (subroutine adapted from that used by Kirkland in [Kirkland 2010]). The kernel is designed in such a way as to loop through all of the atoms in each of the bins which are in close proximity to the location of each thread. This is done in such a way as to ensure each multiprocessor loops through the same group of atoms to allow for broadcast read operations, however this is still suboptimal as it does not use shared memory like the other transmission function kernels below.

```
1  __device__ float vatom( int Z, float radius,const float * devfparams)
2  {
3     float sumlorentz, sumgauss, x, r;
4     r = fabs( radius);
5     r= sqrt(r);
6     if( r < 1e-10f ) r = 1e-10f;
7     sumlorentz = sumgauss = 0.0;
```

```
    x = 2.0*PI*r;

    for(int i=0; i<2*3; i+=2 )
        sumlorentz += devfparams[(Z-1)*12+i]* exp( -x*sqrt(devfparams[(Z-1)*12+i+1]) );

    x = PI*PI*r*r;

    for(int i = 2*3; i < 2*(3+3); i+=2 )
        sumgauss += devfparams[(Z-1)*12+i]*exp(-x/devfparams[(Z-1)*12+i+1])
                 *powf(devfparams[(Z-1)*12+i+1],-1.5f);

    return( (1.0f/r)*150.4121417f*sumlorentz + 266.5157269f*sumgauss );
}
```

This subroutine determines the potential at a distance r from an atom of type z by parametrising the atomic potential as a sum of three Lorentzians and three Gaussians. Each Lorentzian/Gaussian is described by two parameters which are stored in the array devfparams in constant memory. This is based on the method used by Kirkland in [Kirkland 2010].

A.1.3 Transmission Function Code - Improved Multislice

```
__global__ void binnedAtomPKernel3Shared(cuComplex* __restrict__ V, float pixelscale,
    const float* __restrict__ devfparams, const float* __restrict__ devAtomZPos,
    const float* __restrict__ devAtomXPos, const float* __restrict__ devAtomYPos,
    const int* __restrict__ devAtomZNum, const int* __restrict__ devBlockStartPositions,
    float dz, float convdz, float z, int nSlices,float sigma, float blockoffsetx,
    float blockoffsety)
{
    float vatomfast(int Z, float radius);

    float sumz = 0; // To store value of potential, initialized to zero for call of each slice.
    float rad2 = 0;

    int xIndex = threadIdx.x + blockIdx.x * blockDim.x; // ID of thread in grid
    int yIndex = threadIdx.y + blockIdx.y * blockDim.y; // ID of thread in grid
    int lid = threadIdx.x + blockDim.x * threadIdx.y; // ID of thread in workgroup

    __shared__ float atx[256]; // Workgroup specific memory
    __shared__ float aty[256];
    __shared__ float atz[256];
    __shared__ int atZ[256];

    if (xIndex >= res[0] || yIndex >= res[1])
        return ;

    int Index = (yIndex * res[0]) + xIndex;

    // Determine top and bottom slices to include atoms from
    int topzid  = floor((maxs[4]-maxs[5]-z)/dz)-consts[2];
    int bottomzid = floor((maxs[4]-maxs[5]-z)/dz)+consts[2];

    // Determine first and last block to load atoms from (X)
    int si = fmax(floor(((blockoffsetx+blockIdx.x  * blocks[0] * pixelscale)* bins[0]
```

```
33          / ( maxs[0] - maxs[1] )) - consts[0] ),0);
34       int ei = fmin(ceil(((blockoffsetx+(blockIdx.x+1)  * blocks[0] * pixelscale)* bins[0]
35          / ( maxs[0] - maxs[1] )) + consts[0] ),bins[0]-1);
36
37       // Can't include slices that don't exist beyond sample boundary.
38       if (topzid < 0) topzid = 0;
39       if (bottomzid >= nSlices) bottomzid = nSlices-1;
40
41       float rminsq = pixelscale*pixelscale/16.0f;
42
43       // Determine first and last block to load atoms from (Y)
44       int startj = fmax(floor(((blockoffsety+blockIdx.y * blocks[1] * pixelscale)* bins[1]
45          / ( maxs[2] - maxs[3] )) - consts[1] ),0);
46       int endj = fmin(ceil(((blockoffsety+(blockIdx.y+1) * blocks[1] * pixelscale)* bins[1]
47          / ( maxs[2] - maxs[3] )) + consts[1]),bins[1]-1);
48
49       for(int k = topzid; k <= bottomzid; k++)
50       {
51          for (int j = startj; j <= endj;  j++)
52          {
53             // ID of first and last atom required.
54             int start = devBlockStartPositions[k*bins[0]*bins[1] + bins[0]*j + si];
55             int end = devBlockStartPositions[k*bins[0]*bins[1] + bins[0]*j + ei+1];
56
57             int gid = start + lid;
58             // load atoms (one per thread)
59             if(lid < end-start)
60             {
61                atx[lid] = devAtomXPos[gid];
62                aty[lid] = devAtomYPos[gid];
63                atz[lid] = devAtomZPos[gid];
64                atZ[lid] = devAtomZNum[gid];
65             }
66
67             __syncthreads(); // Synchronise across workgroup
68
69             float p2 = 0;
70             for (int l = 0; l < end-start; l++)
71             {
72
73                for(int m = 0; m <= INTEGRALS; m++)
74                {
75                   float z2 = z - m * convdz/INTEGRALS;
76                   rad2 = (blockoffsetx+xIndex*pixelscale-atx[l])*(blockoffsetx+xIndex*pixelscale-atx[l])
77                      + (blockoffsety+yIndex*pixelscale-aty[l])*(blockoffsety+yIndex*pixelscale-aty[l])
78                      + (z2-atz[l])*(z2-atz[l]);
79
80                   if( rad2 < 9.0f) // Check atom is within specified range.
81                   {
82                      // Determine potential at this point
83                      float p1 = vatomfast(atZ[l] , (rad2 < rminsq) ? rminsq : rad2);
84                      sumz += (m!=0) ? (p1+p2) * 0.5f : 0; // Numerical integration of potential
85                      p2 = p1;
86                   }
87                }
88             }
```

```
89            __syncthreads(); // Synchronise across workgroup
90        }
91    }
92
93    V[Index].x = cosf(sigma*convdz/INTEGRALS*sumz); // Calculate transmission function from potential
94    V[Index].y = sinf(sigma*convdz/INTEGRALS*sumz);
95 }
```

This kernel is the fully optimised version of the transmission function calculation for the IMS method. Each multiprocessor allocates 4 arrays in local/shared memory atx,aty,atz,atZ which are used to store the information about the atoms that the multiprocessor will be calculating potential contributions from. The atoms are already divided into groups based on spatial proximity and so a small calculation is done to determine which groups of atoms need to be loaded, the position in memory of the first and last atom to be loaded are determined and then each thread is tasked with loading a single atom into the shared memory arrays, this is only possible because the group of atoms from a row of atom groups are stored consecutively in memory due to the presorting function.

A synchronisation function is then required to ensure that all threads have loaded atoms before and further calculations are done. Each thread then loops through the group of atoms in the shared memory to determine its potential contribution to that thread. As the atoms are in shared memory, these reads are very fast and can be broadcast to all threads in the multiprocessor that are requesting them at the same time, if the data was not stored in shared memory this would not be possible. Another synchronisation is employed to ensure each thread has determined the contribution from every atom before the next group of atoms from the next row or slice of atom groups is loaded. After all featuring atom contributions have been determined the transmission functions is determined from the accumulated potential contributions and stored in the output in global memory.

A.1.4 Imaging Kernel

```
1  __global__ void imagingKernel2(cuComplex * PsiIn, int width, int height, float Cs, float df,
2      float dfa2, float dfa2phi, float dfa3,float dfa3phi,float objap,float wavel, cuComplex * PsiOut,
3      float * k0x, float * k0y, float beta, float delta)
4  {
5
6     int xIndex = threadIdx.x + blockIdx.x * blockDim.x;
7     int yIndex = threadIdx.y + blockIdx.y * blockDim.y;
8
9     if (xIndex >= width || yIndex >= height)
10       return ;
11
12    int Index = (yIndex * width + xIndex);
13
14    float objap2 = (((objap * 0.001f) / wavel ) * (( objap * 0.001f ) / wavel ));
15    float Chi1 = PI * wavel;
```

```
    float Chi2 = 0.5f * Cs * wavel * wavel;
    float k2 = (k0x[xIndex]*k0x[xIndex]) + (k0y[yIndex]*k0y[yIndex]);
    float k =sqrtf(k2);
    float factor = 1.0f*beta*beta/(4*wavel*wavel);

    float ecohs = expf(-factor*pow(PI*k*wavel*2*df + 2*PI*wavel*wavel*wavel*Cs*k2*k,2));
    float ecohd = expf(-0.25f*delta*delta*PI*PI*k2*k2*wavel*wavel);

    if ( k2 < objap2){
       float Phi = atan2(k0y[yIndex],k0x[xIndex]);
       float Chi = Chi1 * k2 * (Chi2 * k2 + df + dfa2 * sin (( 2.0f * (Phi - dfa2phi)))
       + 2.0f * dfa3 * wavel * sqrtf(k2) * sinf (( 3.0f * (Phi - dfa3phi))) / 3.0f);
       PsiOut[Index].x = ecohs*ecohd*(PsiIn[Index].x*  cosf(Chi)  + PsiIn[Index].y * sinf (Chi)) ;
       PsiOut[Index].y   = ecohs*ecohd*(PsiIn[Index].x*- sinf(Chi) + PsiIn[Index].y * cosf (Chi));
    }
    else {
       PsiOut[Index].x = 0.0f;
       PsiOut[Index].y = 0.0f;
    }
}
```

This function takes the exit wave as input and applies the lens transfer function to create the image wavefunction which can then be converted into an image intensity later. The inputs to this functions are the aberration magnitudes and angles which are used to determine the aberration function and size of the coherence envelopes.

A.2 FTSR Functions

The following segments of code are functions designed to be run on the GPU taken from the full exit wave reconstruction plugin which is too large to present here. The kernels used for image alignment as well as calculation restoration filters are presented here along with as supplementary functionality such as the diffractogram fitting routine.

A.2.1 Wave Tranfer Function Calculation Code

```
__kernel void clWaveTransferFunction(__global float2* clw, __global const float* CLxFrequencies,
   __global const float* CLyFrequencies, int sizeX, int sizeY, float wavelength, float beta,
   float delta, float A1r, float A1i, float C1, float C3, float objap)
{
   //Get the work items ID
   int xid = get_global_id(0);
   int yid = get_global_id(1);

   if(xid<sizeX&&yid<sizeY)
   {
      int Index = xid + yid*sizeX;
      float frequency = sqrt(CLxFrequencies[xid]*CLxFrequencies[xid]
         + CLyFrequencies[yid]*CLyFrequencies[yid]);
```

```
        float kx = CLxFrequencies[xid];
        float ky = CLyFrequencies[yid];
        float SpatCohPart = fabs( wavelength * 2 * 3.141592654f * C1 * frequency +
           2 * 3.141592654f * wavelength * wavelength * wavelength * C3
           * frequency * frequency * frequency );
        float CohEnv = exp( -(( beta * beta )/( 4 * wavelength * wavelength ))
           * SpatCohPart * SpatCohPart ) *   exp( -(0.5f * delta * delta)
           *(fabs( 3.141592654f * wavelength * frequency * frequency )
           * fabs( 3.141592654f * wavelength * frequency * frequency ));
        float gamma = ( .166666666f * 3.141592654f * wavelength )
           * ( 6 * A1r * kx * kx + 6 * C1 * kx * kx - 6 * A1r * ky * ky
                + 6 * C1 * ky * ky + 3 * C3 * wavelength * wavelength * kx * kx * kx * kx
                + 6 * C3 * wavelength * wavelength * kx * kx * ky * ky
                + 3 * C3 * wavelength * wavelength * ky * ky * ky * ky + 12 * A1i * kx * ky) ;
        clw[Index].x = (frequency<=objap) * CohEnv * cos(gamma);
        clw[Index].y = (frequency<=objap) * CohEnv * -1 * sin(gamma);
        if(Index==0)
        {
           clw[Index].x = 0.0f;
           clw[Index].y = 0.0f;
        }
    }
}
```

This functions is used to determine the wave transfer function for a given set of aberration parameters. Each pixel in the image represents and individual spatial frequency which have been predetermined and uploaded into GPU memory in the CLxFrequencies and CLyFrequencies arrays which are supplied as inputs.

A.2.2 Restoration Filter Kernels

```
__kernel void clWienerW(__global float* clW, __global float2* clw, int sizeX, int sizeY, int init)
{
   //Get the work items ID
   int xid = get_global_id(0);
   int yid = get_global_id(1);

   if(xid<sizeX&&yid<sizeY)
   {
      int Index = xid + yid*sizeX;
      float absw = sqrt(clw[Index].x*clw[Index].x + clw[Index].y*clw[Index].y);
      if(init==1)
      {
         clW[Index] = 0;
      }
      clW[Index] += absw*absw;
   }
}
```

```
__kernel void clWienerV(__global float2* clV, __global float2* clw, __global float2* clwminus,
   int sizeX, int sizeY, int init)
{
   //Get the work items ID
   int xid = get_global_id(0);
```

```
    int yid = get_global_id(1);

    if(xid<sizeX&&yid<sizeY)
    {
        int Index = xid + yid*sizeX;
        float real = clw[Index].x * clwminus[Index].x - clw[Index].y * clwminus[Index].y;
        float imag = clw[Index].x * clwminus[Index].y + clw[Index].y * clwminus[Index].x;
        if(init==1)
        {
            clV[Index].x = 0;
            clV[Index].y = 0;
        }
        clV[Index].x += real;
        clV[Index].y += imag;
    }
}
```

```
__kernel void clWienerT(__global float2* clT, __global float2* clw, __global float2* fft,
    int sizeX, int sizeY, int init)
{
    //Get the work items ID
    int xid = get_global_id(0);
    int yid = get_global_id(1);

    if(xid<sizeX&&yid<sizeY)
    {
        int Index = xid + yid*sizeX;
        float real = clw[Index].x*fft[Index].x + clw[Index].y*fft[Index].y;
        float imag = clw[Index].x*fft[Index].y - clw[Index].y*fft[Index].x;
        if(init==1)
        {
            clT[Index].x = 0;
            clT[Index].y = 0;
        }
        clT[Index].x += real;
        clT[Index].y += imag;
    }
}
```

```
__kernel void clWienerU(__global float2* clU, __global float2* clwminus,
    __global float2* fft, int sizeX, int sizeY, int init)
{
    //Get the work items ID
    int xid = get_global_id(0);
    int yid = get_global_id(1);

    if(xid<sizeX&&yid<sizeY)
    {
        int Index = xid + yid*sizeX;
        float real = clwminus[Index].x*fft[Index].x - clwminus[Index].y*fft[Index].y;
        float imag = clwminus[Index].x*fft[Index].y + clwminus[Index].y*fft[Index].x;
        if(init==1)
        {
            clU[Index].x = 0;
            clU[Index].y = 0;
        }
        clU[Index].x += real;
```

```
        clU[Index].y += imag;
    }
}
```

These 4 kernels are used to calculate the 5 running sums (first one used twice) used to determine the restored wavefunction. They are each launched multiple times, (once per image in the reconstruction) along with the wave transfer function determined for the corresponding image. When launched with the appropriate flag the kernel will set the initialized value to zero so that this does not have to be performed on the CPU which would be less efficient.

```
__kernel void clMakeRestored(__global float* clW,__global float* clWminus, __global float2* clV,
    __global float2* clT, __global float2* clU, __global float2* clRestored, int sizeX,
    int sizeY, float snr)
{
    //Get the work items ID
    int xid = get_global_id(0);
    int yid = get_global_id(1);

    if(xid<sizeX&&yid<sizeY)
    {
        int Index = xid + yid*sizeX;
        float real = (clWminus[Index] + snr)*clT[Index].x
            - clV[Index].x * clU[Index].x - clV[Index].y * clU[Index].y;
        float imag = (clWminus[Index] + snr)*clT[Index].y
            - clV[Index].x * clU[Index].y + clV[Index].y * clU[Index].x;
        float denom = ((clWminus[Index] + snr) * (clW[Index] + snr))
            - ( clV[Index].x * clV[Index].x + clV[Index].y * clV[Index].y );
        clRestored[Index].x = real/denom;
        clRestored[Index].y = imag/denom;
    }
}
```

This kernel takes the 5 running sums calculated using the above kernels and uses them to generate the reconstructed wavefunction.

A.2.3 PCI Determination

```
__kernel void clCalculatePCI(__global float2* clQ, __global float2* clFFT,
    __global float2* clFFTminus, __global float* CLxFrequencies, __global float* CLyFrequencies,
    __global float2* clPCI, __global float2* clPCIC, __global float2* clPCIM,
    __global float2* clPCIS, int sizeX, int sizeY, float C1, float C3, float A1r,
    float A1i, float wavelength, float objap)
{
    //Get the work items ID
    int xid = get_global_id(0);
    int yid = get_global_id(1);

    if(xid<sizeX&&yid<sizeY)
    {
        int Index = xid + yid*sizeX;
        float freq = sqrt(CLxFrequencies[xid]*CLxFrequencies[xid]
```

```
15          + CLyFrequencies[yid]*CLyFrequencies[yid]);
16       float kx = CLxFrequencies[xid];
17       float ky = CLyFrequencies[yid];
18       float gamma = ( .166666666 * 3.1415927f * wavelength ) * ( 6 * A1r * kx * kx + 6 * C1 * kx * kx
19           - 6 * A1r * ky * ky   + 6 * C1 * ky * ky + 3 * C3 * wavelength * wavelength * kx * kx * kx * kx
20          + 6 * C3 * wavelength * wavelength * kx * kx * ky * ky
21          + 3 * C3 * wavelength * wavelength * ky * ky * ky * ky
22          + 12 * A1i * kx * ky) ;
23       float arg1 = atan2(clFFT[Index].y,clFFT[Index].x);
24       float arg2 = atan2(clFFTminus[Index].y,clFFTminus[Index].x);
25       float angle = atan2(ky,kx);
26       clPCI[Index].x = (freq<=objap)*clQ[Index].x*-1 *cos(arg1+arg2+2*gamma);
27       clPCI[Index].y = 0;
28       clPCIC[Index].x = (freq<=objap)*clQ[Index].x*-1 *cos(arg1+arg2+2*gamma)*cos(2*angle);
29       clPCIC[Index].y = 0;
30       clPCIS[Index].x = (freq<=objap)*clQ[Index].x*-1 *cos(arg1+arg2+2*gamma)*sin(2*angle);
31       clPCIS[Index].y = 0;
32       clPCIM[Index].x = (freq<=objap)*(clPCI[Index].x + sqrt(clPCIC[Index].x * clPCIC[Index].x
33          + clPCIS[Index].x * clPCIS[Index].x));
34       clPCIM[Index].y = 0;
35    }
36 }
```

This kernel is used to aid in determination of the residual defocus and astigmatism in a reconstructed wavefunction. The kernel is launched once per defocus/astigmatism combination and produces 4 arrays with the same dimensions as the input wavefunction (clPCI/C/S/M) as outputs. These arrays are then summed using another GPU kernel to produce a single number representing the likelihood that this is the correct defocus and astigmatism. As this entire procedure is repeated once per trial defocus value, this is a very expensive part of the calculation procedure.

A.3 Image Alignment

A.3.1 PCPCF Calculation Code

```
1  __kernel void clPCPCF(__global const float2* fft1, __global const float2* fft2,
2     __global float2* fftresult,   __global const float* CLxFrequencies,
3     __global const float* CLyFrequencies, int sizeX, int sizeY,
4       float focalstep, float wavelength, float pcpcfkmax)
5  {
6     //Get the work items ID
7     int xid = get_global_id(0);
8     int yid = get_global_id(1);
9     if(xid<sizeX&&yid<sizeY)
10    {
11       int Index = xid + yid*sizeX;
12       float frequency = sqrt(CLxFrequencies[xid]*CLxFrequencies[xid]
13                      + CLyFrequencies[yid]*CLyFrequencies[yid]);
14       float compensation = cos(3.1415926534f * focalstep * wavelength * frequency * frequency);
15       float c1r = fft1[Index].x;
16       float c1i = fft1[Index].y;
17       float c2r = fft2[Index].x;
```

```
        float c2i = fft2[Index].y;
        float denom2 = sqrt( compensation*compensation   * (c1r*c1r*c2r*c2r + c1i*c1i*c2i*c2i
                    + c1r*c1r*c2i*c2i + c2r*c2r*c1i*c1i) + 0.001 * 0.001
                    + 2 * 0.001 * compensation * (c1r*c2r + c1i*c2i));
        fftresult[Index].x = (frequency<=pcpcfkmax)*compensation*(c1r*c2r + c1i*c2i)/denom2;
        fftresult[Index].y = (frequency<=pcpcfkmax)*compensation*(c2i*c1r - c1i*c2r)/denom2;
    }
}
```

This GPU kernel calculates the PCPCF for two input image Fourier transforms and a given defocus and wavelength. Each pixel is calculated in a separate thread. The spatial frequencies k_x, k_y are pre-calculated and retrieved from the GPUs global memory instead of calculating the spatial frequency for each pixel within the kernel itself as these values are reused in a large number of kernels.

A.3.2 Mutual Information GPU Kernel Code

```
__kernel void clJointHistogramMULTI(__global const float* ImageData1,
    __global const float* ImageData2, __global int* JointHistograms, int sizeX,
    int sizeY, float max, float min, float max2, float min2, int xs, int ys,
    __local int* tmp_histogram, __local int* tmp_histogram2)
{
    //Get the work items ID
    int xid = get_global_id(0);
    int yid = get_global_id(1);
  // Read bin values into appropriate part of shared memory in parallel
    for(int lidx=  get_local_id(0); lidx < (get_local_size(0)+19); lidx += get_local_size(0))
        for(int lidy=  get_local_id(1); lidy < (get_local_size(1)+19); lidy += get_local_size(1))
        {
            int xp = lidx + get_group_id(0)*get_local_size(0) + xs;
            int yp = lidy + get_group_id(1)*get_local_size(1) + ys;
            int x2 = lidx + get_group_id(0)*get_local_size(0);
            int y2 = lidy + get_group_id(1)*get_local_size(1);
            if(xp>=sizeX)
                xp-=sizeX;
            if(yp>=sizeY)
                yp-=sizeY;
            if(xp<0)
                xp+=sizeX;
            if(yp<0)
                yp+=sizeY;
            if(x2>=sizeX) // Not necessary, don't use extra
                x2-=sizeX;
            if(y2>=sizeY) // Not necessary, don't use extra
                y2-=sizeY;
            // Want normal bin values from first image, but the second image
            // should also take into account the xs and ys so we can fit
            // the full range into multiple kernel launches
            tmp_histogram[lidx + (get_local_size(0)+19)*lidy]
                = floor((ImageData1[x2 + sizeX * y2] - min)/(max-min) * 255.0f);
            tmp_histogram2[lidx + (get_local_size(0)+19)*lidy]
                = floor((ImageData2[xp + sizeX * yp] - min2)/(max2-min2) * 255.0f);
        }
    barrier(CLK_LOCAL_MEM_FENCE)
```

```
38   // At this point have a shared memory full of B1 and B2 values to make a
39   // 20*20 set of Joint Histograms for every pixel in this workgroup
40     for( int xx = 0; xx < 20; xx++)
41        for( int yy = 0; yy < 20; yy++)
42        {
43           int histo = xx + 20*yy;
44           int b1= tmp_histogram[get_local_id(0)+get_local_id(1)*(get_local_size(0)+19)];
45           int b2= tmp_histogram2[get_local_id(0)+xx+(get_local_id(1)+yy)*(get_local_size(0)+19)];
46           if(b1>255) // Can have bins outside correct range as it was based on contrast limits.
47              b1=255;
48           if(b2>255)
49              b2=255;
50           if(b1<0)
51              b1=0;
52           if(b2<0)
53              b2=0;
54           // Increment using atomics to prevent data race.
55           atomic_inc(&JointHistograms[(histo*256*256) + b1 + (256 * b2)]);
56        }
57   }
```

This kernel calculates a joint histogram from two input images for 20×20 possible image displacements at the same time. An additional offset is input so that a different 20×20 grid of image displacements can be calculated in a subsequent kernel launch. The kernel makes significant use of shared memory to store histogram bin values so that they can be accessed by other threads later in the computation to save reading values from the original image again which is much slower as the original image is stored in global memory.

Each thread originally determines the histogram bin for a single pixel from each of the two images and stores this value in shared memory (some threads will calculate two pixels to cover an overlap region). Synchronisation is then used to ensure each thread has saved its bin value before any of the values are subsequently retrieved in another thread which could otherwise be at a different point in the calculation.

The threads then loop through the saved histogram bin values for the second image and increment the correct bin in the correct joint histogram without needing to reload the images or redetermine the histogram bins. Histograms must be updated using atomic operations to prevent multiple threads incrementing a bin at the same time and causing data races.

A.4 Supplementary Functions

A.4.1 Diffractogram Matching Code

```
1  __kernel void clDiffMatch(__global const float* quadrantData, __global float* result,
2     __local float* localQuadrant, int quadrant, int width, int height, int depth,
3     int quadrantwidth, int quadrantheight, float dfstart, float dfstep,
```

```
    float astigxstart, float astigxstep, float astigystart, float astigystep,
    float pixelscale, float wavelength, float Cs, float noincluderegion)
{
    int xid = get_global_id(0);
    int yid = get_global_id(1);
    int zid = get_global_id(2);

    int times = ( quadrantwidth * quadrantheight + ( get_local_size(0) *
       get_local_size(1) ) - 1 ) / ( get_local_size(0) * get_local_size(1) );
    if(get_local_id(0) < quadrantwidth && get_local_id(1) < quadrantheight
       && get_local_id(2)== 0) {
       for(int i = 0 ; i < times ; i++)
          if(get_local_id(0) + get_local_id(1)*get_local_size(0) + i*get_local_size(0)
                *get_local_size(1) < quadrantwidth*quadrantheight) {
             localQuadrant[get_local_id(0) + get_local_id(1)*get_local_size(0)
             + i*get_local_size(0)*get_local_size(1)]
             = quadrantData[get_local_id(0) + get_local_id(1)*get_local_size(0)
                + i*get_local_size(0)*get_local_size(1)];
          }
    }
    barrier(CLK_LOCAL_MEM_FENCE);
    if(xid >= width || yid >= height || zid >= depth)
       return;
    float defocus = dfstart + xid*dfstep;
    float astigx = astigxstart + yid*astigxstep;
    float astigy = astigystart + zid*astigystep;
    float astigma = sqrt(astigx*astigx + astigy*astigy);
    float astigaxis = atan2(astigy+0.000001f,astigx+0.000001f);
    float s1 = 0.0f;
    float s2 = 0.0f;
    float s3 = 0.0f;
    int quadi = 1;
    int quadj = 1;
    if(quadrant==3||quadrant==4)
       quadi = -1;
    if(quadrant==1||quadrant==4)
       quadj = -1;
    for(int i = 1; i <= quadrantwidth; i++)
       for(int j = 1; j<= quadrantheight; j++) {
          if(((quadrantwidth-i)*(quadrantwidth-i) +
              (quadrantheight-j)*(quadrantheight-j)) >
             noincluderegion*noincluderegion) {
             if(localQuadrant[(i-1)+quadrantwidth*(j-1)]*
                localQuadrant[(i-1)+quadrantwidth*(j-1)]!=0) {
                float k = sqrt((quadrantwidth-i)*(quadrantwidth-i)*pixelscale*pixelscale +
                        (quadrantheight-j)*(quadrantwidth-j)*pixelscale*pixelscale);
                float val = -sin(2*3.1415926f*((wavelength/2.0f)*defocus*k*k +
                          (wavelength/2.0f)*astigma*k*k*cos(2.0f*(atan2(quadj*
                          (quadrantheight+0.001f - j), quadi*(quadrantwidth+0.01f - i)) -
                           astigaxis)) + ((wavelength*wavelength*wavelength)/4)*Cs*k*k*k*k));
                float val2 = val*val;
                s1 += (val2 * localQuadrant[(i-1)+quadrantwidth*(j-1)]);
                s2 +=  val2*val2;
                s3 +=  localQuadrant[(i-1)+quadrantwidth*(j-1)]*
                      localQuadrant[(i-1)+quadrantwidth*(j-1)];
             }
```

```
60          }
61       }
62
63    result[xid + yid * width + zid * width * height] = s1/sqrt(s2*s3);
64 }
```

This kernel is used to simulate diffractograms for a range of imaging conditions and also to determine how accurate these simulations are to a supplied reference diffractogram. Each thread uses its ID to determine which imaging conditions to simulate for, then simulates its own separate diffractogram and calculates the difference between it and the reference. As the simulation and comparison are carried out at the same time, the diffractograms do not have to be stored which would be impossible for the number of diffractograms simulated concurrently.

A.4.2 Maximum/Total Reduction Kernels

```
1  __kernel void clMaxReduction(__global const float* input, __global float* output,
2        const unsigned int size, __local float* buffer)
3  {
4     //Get the work items ID
5     size_t idx = get_local_id(0);
6     size_t stride = get_global_size(0);
7     buffer[idx] = 0;
8
9     // Copy data into shared memory buffer, each threads gets
10    // multiple values if we launch less threads than size
11    for(size_t pos = get_global_id(0); pos < size; pos += stride )
12       buffer[idx] = max(buffer[pos],buffer[idx]);
13    barrier(CLK_LOCAL_MEM_FENCE);
14
15    float max = 0;
16
17    // Loop over all values with first thread in workgroup
18    if(!idx) {
19       for(size_t i = 1; i < get_local_size(0); ++i)
20          max = max(sum,buffer[i]);
21       output[get_group_id(0)] = max;
22    }
23 }
```

This kernel is used to aid in determining the maximum value in a large array by finding the maximum value in several subsets of the array in parallel and returning these values. This leaves a much smaller array where the maximum can quickly be determined via CPU.

```
1  __kernel void clMaxReduction(__global float2* restrict buffer,
2              __local float* scratch,
3              __local uint* scratchpos,
4              __const int length,
5              __global float* restrict result,
6              __global uint* position) {
```

```
7
8     int global_index = get_global_id(0);
9     float accumulator = -10000000.0f; // Small number
10    uint pos = global_index;
11    // Loop sequentially over chunks of input vector
12    while (global_index < length) {
13      float element = buffer[global_index].x;
14      accumulator = (accumulator > element) ? accumulator : element;
15      pos = (accumulator > element) ? pos : global_index;
16      global_index += get_global_size(0);
17    }
18
19    // Perform parallel reduction
20    int local_index = get_local_id(0);
21    scratch[local_index] = accumulator;
22    scratchpos[local_index] = pos;
23    barrier(CLK_LOCAL_MEM_FENCE);
24    for(int offset = get_local_size(0) / 2;
25        offset > 0;
26        offset = offset / 2) {
27      if (local_index < offset) {
28        float other = scratch[local_index + offset];
29        float mine = scratch[local_index];
30        scratch[local_index] = (mine > other) ? mine : other;
31        scratchpos[local_index] = (mine > other) ? scratchpos[local_index] : scratchpos[local_index + offset];
32      }
33      barrier(CLK_LOCAL_MEM_FENCE);
34    }
35    if (local_index == 0) {
36      result[get_group_id(0)] = scratch[0];
37      position[get_group_id(0)] = scratchpos[0];
38    }
39  }
```

This kernel is somewhat similar to the previous maximum value reduction kernel, however the reductions are not all performed in the first thread of the workgroup, instead they are performed on 1 of every 2^N threads reducing the number of possible maxima by a factor of two until only one final value is left for each workgroup. Intermediary results are also stored in local memory for efficiency. This kernel also stores the position of the maximum values within the array, this is useful for image alignment where the position of the maximum value is used to derive the image displacement. [1]

```
1   __kernel void clSumReduction(__global const float* input,
2      __global float* output, const unsigned int size,
3      __local float* buffer, int offset)
4   {
5      //Get the work items ID
6      size_t idx = get_local_id(0);
```

[1] This kernel has been adapted from an AMD example kernel given at http://developer.amd.com/resources/documentation-articles/articles-whitepapers/opencl-optimization-case-study-simple-reductions/

```
7     size_t stride = get_global_size(0);
8     buffer[idx] = 0;
9
10    for(size_t pos = get_global_id(0); pos < size; pos += stride )
11       buffer[idx] += input[offset+pos];
12    barrier(CLK_LOCAL_MEM_FENCE);
13
14    float sum = 0;
15
16    // Loop over all values with the first thread in the workgroup
17    if(!idx) {
18       for(size_t i = 1; i < get_local_size(0); ++i)
19          sum += buffer[i];
20       output[get_group_id(0)] = sum;
21    }
22 }
```

This kernel is used to aid in determining the total of the values in a large array by finding the total of several subsets of the array in parallel and returning these values. This leaves a much smaller array where the total can quickly be determined via CPU.

Bibliography

Advanced Micro Devices Inc.,. AMD Accelerated Parallel Processing OpenCL Programming Guide, 2013. URL `http://developer.amd.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf`.

Allen, L. J., McBride, W., O'Leary, N. L., and Oxley, M. P. Exit wave reconstruction at atomic resolution. *Ultramicroscopy*, 100(1-2):91–104, July 2004. doi:10.1016/j.ultramic.2004.01.012.

Anstis, G. The calculation of electron diffraction intensities by the multislice method. *Acta Crystallogr. Sect. A*, 33(5):844–846, September 1977.

Avouris, P. and Freitag, M. Graphene Photonics, Plasmonics, and Optoelectronics, 2014.

Barthel, J. and Thust, a. Aberration measurement in HRTEM: implementation and diagnostic use of numerical procedures for the highly precise recognition of diffractogram patterns. *Ultramicroscopy*, 111(1):27–46, December 2010. doi:10.1016/j.ultramic.2010.09.007.

Bethe, H. Theorie der Beugung von Elektronen an Kristallen. *Ann. Phys.*, 392(17): 55–129, January 1928. doi:10.1002/andp.19283921704.

Birk, M., Dapp, R., Ruiter, N. V., and Becker, J. GPU-based iterative transmission reconstruction in 3D ultrasound computer tomography. *J. Parallel Distrib. Comput.*, 74(1):1730–1743, January 2014. doi:http://dx.doi.org/10.1016/j.jpdc.2013.09.007.

Bonaccorso, F., Sun, Z., Hasan, T., and Ferrari, a. C. Graphene photonics and optoelectronics. *Nat. Photonics*, 4(9):611–622, August 2010. doi:10.1038/nphoton.2010.186.

Brown, L. A survey of image registration techniques. *ACM Comput. Surv.*, pages 1–60, 1992.

Buist, A. H., van den Bos, A., and Miedema, M. A. O. Optimal experimental design for exit wave reconstruction from focal series in TEM. *Ultramicroscopy*, 64(1âĂŞ4): 137–152, August 1996a. doi:http://dx.doi.org/10.1016/0304-3991(96)00016-2.

Buist, A., Bos, A. V. D., and Miedema, M. Optimal experimental design for exit wave reconstruction from focal series in TEM. *Ultramicroscopy*, 64:137–152, 1996b.

Cahill, N. Normalized measures of mutual information with general definitions of entropy for multimodal image registration. *Biomed. Image Regist.*, (8):258–268, 2010.

Cai, C. and Chen, J. An accurate multislice method for low-energy transmission electron microscopy. *Micron*, 43(2âĂŞ3):374–379, February 2012. doi:http://dx.doi.org/10.1016/j.micron.2011.09.018.

Chang, W.-H., Chiu, M. T. K., Chen, C.-Y., Yen, C.-F., Lin, Y.-C., Weng, Y.-P., Chang, J.-C., Wu, Y.-M., Cheng, H., Fu, J., and Tu, I. P. Zernike Phase Plate Cryoelectron Microscopy Facilitates Single Particle Analysis of Unstained Asymmetric Protein Complexes. *Structure*, 18(1):17–27, 2010.

Coene, W. M. J., Thust, A., Op de Beeck, M., and Van Dyck, D. Maximum-likelihood method for focus-variation image reconstruction in high resolution transmission electron microscopy. *Ultramicroscopy*, 64(1âĂŞ4):109–135, 1996. doi:10.1016/0304-3991(96)00010-1.

Coene, W., Janssen, G., Op de Beeck, M., and Van Dyck, D. Phase retrieval through focus variation for ultra-resolution in field-emission transmission electron microscopy. *Phys. Rev. Lett.*, 69(26):3743–3746, December 1992.

Cooley, J. and Tukey, J. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, pages 297–301, 1965.

Cowley, J. M. and Moodie, a. F. The scattering of electrons by atoms and crystals. I. A new theoretical approach. *Acta Crystallogr.*, 10(10):609–619, October 1957. doi:10.1107/S0365110X57002194.

Croitoru, M. D., Van Dyck, D., Van Aert, S., Bals, S., and Verbeeck, J. An efficient way of including thermal diffuse scattering in simulation of scanning transmission electron microscopic images. *Ultramicroscopy*, 106(10):933–40, 2006. doi:10.1016/j.ultramic.2006.04.006.

DAlfonso, A. J., Morgan, A. J., Martin, A. V., Quiney, H. M., and Allen, L. J. Fast deterministic approach to exit-wave reconstruction. *Phys. Rev. A*, 85(1):13816, January 2012.

D'Alfonso, A. J., Morgan, A. J., Yan, A. W. C., Wang, P., Sawada, H., Kirkland, A. I., and Allen, L. J. Deterministic electron ptychography at atomic resolution. *Phys. Rev. B*, 89(6):64101, February 2014.

Danev, R. and Nagayama, K. Transmission electron microscopy with Zernike phase plate. 88(September 2000):243–252, 2001.

Danev, R., Glaeser, R. M., and Nagayama, K. Practical factors affecting the performance of a thin-film phase plate for transmission electron microscopy. *Ultramicroscopy*, 109:312–325, 2009. doi:10.1016/j.ultramic.2008.12.006.

David, L., Bhandavat, R., and Singh, G. MoS2/Graphene Composite Paper for Sodium-Ion Battery Electrodes. *ACS Nano*, 8(2):1759–1770, February 2014. doi:10.1021/nn406156b.

Doyle, P. a. and Turner, P. S. Relativistic HartreeâĂŞFock X-ray and electron scattering factors. *Acta Crystallogr. Sect. A*, 24(3):390–397, May 1968. doi:10.1107/S0567739468000756.

Erni, R. *Aberration-Corrected Imaging in Transmission Electron Microscopy: An Introduction*. Imperial College Press, London, 2010. ISBN 1-84816-536-6.

Erni, R., Rossell, M. D., and Nakashima, P. N. H. Optimization of exit-plane waves restored from HRTEM through-focal series. *Ultramicroscopy*, 110(2):151–61, January 2010. doi:10.1016/j.ultramic.2009.10.015.

Fujita, T. and McCartney, M. R. Phase recovery for electron holography using GerchbergâĂŞPapoulis iterative algorithm. *Ultramicroscopy*, 102(4):279–286, March 2005. doi:http://dx.doi.org/10.1016/j.ultramic.2004.10.009.

Gabor, D. A New Microscopic Principle. *Nature*, 161(4098):777–778, May 1948. doi:10.1038/161777a0.

Galvão, R. P., Autreto, P. A. S., Legoas, S. B., Srinivasan, S. G., van Duin, A. C. T., and S, D. Graphene to fluorographene and fluorographane: a theoretical study. *Nanotechnology*, 24(3):35706, 2013.

Gamm, B., Dries, M., Schultheiss, K., Blank, H., Rosenauer, a., Schröder, R. R., and Gerthsen, D. Object wave reconstruction by phase-plate transmission electron microscopy. *Ultramicroscopy*, 110:807–814, 2010. doi:10.1016/j.ultramic.2010.02.006.

Genovese, L., Ospici, M., Deutsch, T., Mehaut, J.-F., Neelov, A., and Goedecker, S. Density functional theory calculation on many-cores hybrid central processing unit-graphic processing unit architectures. *J. Chem. Phys.*, 131(3):34103, July 2009. doi:10.1063/1.3166140.

Gerchberg, R. W. A practical algorithm for the determination of phase from image and diffraction plane pictures. *Optik (Stuttg).*, 35:237, 1972.

Goodman, P. and Moodie, A. F. Numerical evaluations of N-beam wave functions in electron scattering by the multi-slice method. *Acta Crystallogr. Sect. A Cryst. Physics, Diffraction, Theor. Gen. Crystallogr.*, 30(2):280–290, 1974.

Götz, A. W., Williamson, M. J., Xu, D., Poole, D., Le Grand, S., and Walker, R. C. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 1. Generalized Born. *J. Chem. Theory Comput.*, 8(5):1542–1555, March 2012. doi:10.1021/ct200909j.

Guerrero, E., Yáñez, A., Galindo, P., Pizarro, J., and Molina, S. I. Influence of atomic displacements due to elastic strain in HAADF-STEM simulated images. In Luysberg, M., Tillmann, K., and Weirich, T., editors, *EMC 2008 14th Eur. Microsc. Congr. 1âĂŞ5 Sept. 2008, Aachen, Ger. SE - 58*, pages 115–116. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-85154-7. doi:10.1007/978-3-540-85156-1_58.

Guerrero-Lebrero, M. P., Pizarro, J., Guerrero, E., Galindo, P. L., Yánez, A., and Molina, S. I. Through-focal HAADF-STEM of buried nanostructures. *J. Phys. Conf. Ser.*, 209:012032, February 2010. doi:10.1088/1742-6596/209/1/012032.

Guizar-Sicairos, M., Thurman, S. T., and Fienup, J. R. Efficient subpixel image registration algorithms. *Opt. Lett.*, 33(2):156–8, January 2008.

Haider, M., Hartel, P., Müller, H., Uhlemann, S., and Zach, J. Current and future aberration correctors for the improvement of resolution in electron microscopy. *Philos. Trans. A. Math. Phys. Eng. Sci.*, 367(1903):3665–82, September 2009. doi:10.1098/rsta.2009.0121.

Hall, R. J., Nogales, E., and Glaeser, R. M. Accurate modeling of single-particle cryo-EM images quantitates the benefits expected from using Zernike phase contrast. *J. Struct. Biol.*, 174(3):468–75, June 2011. doi:10.1016/j.jsb.2011.03.020.

Harris, J. R. and Scheffler, D. Routine preparation of air-dried negatively stained and unstained specimens on holey carbon support films: a review of applications. *Micron*, 33(5):461–480, 2002. doi:10.1016/s0968-4328(01)00039-7.

Herrmann, K.-H. and Krahl, D. The detection quantum efficiency of electronic image recording systems. *J. Microsc.*, 127(1):17–28, July 1982. doi:10.1111/j.1365-2818.1982.tb00393.x.

Hicks, J., Tejeda, a., Taleb-Ibrahimi, a., Nevius, M. S., Wang, F., Shepperd, K., Palmer, J., Bertran, F., Le Fèvre, P., Kunc, J., de Heer, W. a., Berger, C., and

Conrad, E. H. A wide-bandgap metal-semiconductor-metal nanostructure made entirely from graphene. *Nat. Phys.*, 9(1):49–54, 2013. doi:10.1038/nphys2487.

Hosokawa, F., Sawada, H., Kondo, Y., Takayanagi, K., and Suenaga, K. Development of Cs and Cc correctors for transmission electron microscopy. *Reprod. Syst. Sex. Disord.*, 62(1):23–41, February 2013. doi:10.1093/jmicro/dfs134.

Hsieh, W.-K., Chen, F.-R., Kai, J.-J., and Kirkland, a. I. Resolution extension and exit wave reconstruction in complex HREM. *Ultramicroscopy*, 98(2-4):99–114, January 2004. doi:10.1016/j.ultramic.2003.08.004.

Huang, C., Borisenko, K. B., and Kirkland, a. I. Exit wave reconstruction of radiation-sensitive materials from low-dose data. *J. Phys. Conf. Ser.*, 522:012052, 2014. doi:10.1088/1742-6596/522/1/012052.

Hÿtch, M. J. and Stobbs, W. M. Quantitative comparison of high resolution TEM images with image simulations. *Ultramicroscopy*, 53:191–203, 1994. doi:10.1016/0304-3991(94)90034-5.

Inagaki, M. and Kang, F. Graphene derivatives: graphane, fluorographene, graphene oxide, graphyne and graphdiyne. *J. Mater. Chem. A*, 2(33):13193–13206, 2014. doi:10.1039/C4TA01183J.

Ishizuka, K. and Uyeda, N. A new theoretical and practical approach to the multislice method. *Acta Crystallogr. Sect. A*, 2:740–749, 1977.

Ishizuka, K. and Allman, B. Phase measurement of atomic resolution image using transport of intensity equation. *J. Electron Microsc. (Tokyo).*, 54(3):191–7, June 2005. doi:10.1093/jmicro/dfi024.

Jinschek, J. R., Yucelen, E., Calderon, H. A., and Freitag, B. Quantitative atomic 3-D imaging of single/double sheet graphene structure. *Carbon N. Y.*, 49(2): 556–562, February 2011. doi:http://dx.doi.org/10.1016/j.carbon.2010.09.058.

Kirk, D. and Wen-mei, W. *Programming massively parallel processors: a hands-on approach*. Elsevier, Amsterdam, 2012. ISBN 978-0-12-381472-2.

Kirkland, A. I., Saxton, W. O., Chau, K.-L., Tsuno, K., and Kawasaki, M. Super-resolution by aperture synthesis: tilt series reconstruction in CTEM. *Ultramicroscopy*, 57(4):355–374, March 1995. doi:http://dx.doi.org/10.1016/0304-3991(94)00191-O.

Kirkland, E. J. Improved high resolution image processing of bright field electron micrographs: I. Theory. *Ultramicroscopy*, 15(3):151–172, 1984.

Kirkland, E. *Advanced Computing in Electron Microscopy*. Springer, New York, 2nd edition, 2010. ISBN 978-1441965325.

Knoll, M. and Ruska, E. Das elektronenmikroskop. *Zeitschrift für Phys.*, 78:318–339, 1932.

Koch, C. *Determination of core structure periodicity and point defect density along dislocations*. PhD thesis, Arizona State University, 2002.

Koch, C. T. A flux-preserving non-linear inline holography reconstruction algorithm for partially coherent electrons. *Ultramicroscopy*, 108(2):141–50, January 2008. doi:10.1016/j.ultramic.2007.03.007.

Kübel, C. and Thust, A. TrueImage. In Weirich, T., Lábár, J., and Zou, X., editors, *Electron Crystallogr. SE - 23*, volume 211 of *NATO Science Series II: Mathematics, Physics and Chemistry*, pages 373–392. Springer Netherlands, 2006. ISBN 978-1-4020-3918-8. doi:10.1007/1-4020-3920-4_23.

Kurasch, S., Meyer, J. C., Künzel, D., Groß, A., and Kaiser, U. Simulation of bonding effects in HRTEM images of light element materials. *Beilstein J. Nanotechnol.*, 2: 394–404, January 2011. doi:10.3762/bjnano.2.45.

Lentzen, M. Progress in aberration-corrected high-resolution transmission electron microscopy using hardware aberration correction. *Microsc. Microanal.*, 12(3): 191–205, June 2006. doi:10.1017/S1431927606060326.

Lewis, J. Fast normalized cross-correlation. *Vis. interface*, 1995(1), 1995.

Lichte, H. Electron holography approaching atomic resolution. *Ultramicroscopy*, 20: 293–304, 1986.

Lubk, A., Röder, F., Niermann, T., Gatel, C., Joulie, S., Houdellier, F., Magén, C., and Hÿtch, M. J. A new linear transfer theory and characterization method for image detectors. Part II: experiment. *Ultramicroscopy*, 115:78–87, April 2012. doi:10.1016/j.ultramic.2012.01.011.

Meng, Y., Zhao, Y., Hu, C., Cheng, H., Hu, Y., Zhang, Z., Shi, G., and Qu, L. All-Graphene Core-Sheath Microfibers for All-Solid-State, Stretchable Fibriform Supercapacitors and Wearable Electronic Textiles. *Adv. Mater.*, 25(16):2326–2331, 2013. doi:10.1002/adma.201300132.

Meyer, J. C., Kurasch, S., Park, H. J., Skakalova, V., Künzel, D., Gross, A., Chuvilin, A., Algara-Siller, G., Roth, S., Iwasaki, T., Starke, U., Smet, J. H., and Kaiser, U.

Experimental analysis of charge redistribution due to chemical bonding by high-resolution transmission electron microscopy. *Nat. Mater.*, 10(3):209–15, March 2011. doi:10.1038/nmat2941.

Meyer, R. R. *Quantitative Automated Object Wave Restoration in High-resolution Electron Microscopy*. PhD thesis, 2002.

Meyer, R. R., Kirkland, A. I., and Saxton, W. O. A new method for the determination of the wave aberration function for high resolution TEM: 1. Measurement of the symmetric aberrations. *Ultramicroscopy*, 92(2):89–109, 2002. doi:10.1016/s0304-3991(02)00071-2.

Meyer, R. R., Kirkland, a. I., and Saxton, W. O. A new method for the determination of the wave aberration function for high-resolution TEM.; 2. Measurement of the antisymmetric aberrations. *Ultramicroscopy*, 99(2-3):115–23, May 2004. doi:10.1016/j.ultramic.2003.11.001.

Ming, W. Q. and Chen, J. H. Validities of three multislice algorithms for quantitative low-energy transmission electron microscopy. *Ultramicroscopy*, 134:135–43, November 2013. doi:10.1016/j.ultramic.2013.04.013.

Moore, G. E. Cramming More Components onto Integrated Circuits. *Electronics*, 38 (8):114–117, April 1965. doi:doi: 10.1109/jproc.1998.658762.

Morgan, A. J., DâĂŹAlfonso, A. J., Wang, P., Sawada, H., Kirkland, A. I., and Allen, L. J. Fast deterministic single-exposure coherent diffractive imaging at sub-\AA{}ngstr\"om resolution. *Phys. Rev. B*, 87(9):94115, March 2013.

Murata, K., Liu, X., Danev, R., Jakana, J., Schmid, M. F., King, J., Nagayama, K., and Chiu, W. Zernike Phase Contrast Cryo-Electron Microscopy and Tomography for Structure Determination at Nanometer and Subnanometer Resolutions. *Structure*, 18(8):903–912, 2010.

Nagayama, K. and Danev, R. Phase contrast electron microscopy: development of thin-film phase plates and biological applications. *Philos. Trans. R. Soc. Lond. B. Biol. Sci.*, 363(1500):2153–62, July 2008. doi:10.1098/rstb.2008.2268.

Nakashima, P. N. H. and Johnson, a. W. S. Measuring the PSF from aperture images of arbitrary shape–an algorithm. *Ultramicroscopy*, 94(2):135–48, 2003.

Novoselov, K. S., Geim, a. K., Morozov, S. V., Jiang, D., Zhang, Y., Dubonos, S. V., Grigorieva, I. V., and Firsov, a. a. Electric field effect in atomically thin carbon films. *Science*, 306(5696):666–9, October 2004. doi:10.1126/science.1102896.

NVIDIA Corporation,. CUDA C Programming Guide, 2014. URL `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`.

Op de Beeck, M., Van Dyck, D., and Coene, W. Wave function reconstruction in HRTEM: the parabola method. *Ultramicroscopy*, 64(1-4):167–183, August 1996. doi:10.1016/0304-3991(96)00058-7.

Pantelic, R. S., Meyer, J. C., Kaiser, U., Baumeister, W., and Plitzko, J. M. Graphene oxide: A substrate for optimizing preparations of frozen-hydrated samples. *J. Struct. Biol.*, 170(1):152–156, 2010. doi:10.1016/j.jsb.2009.12.020.

Pantelic, R. S., Suk, J. W., Magnuson, C. W., Meyer, J. C., Wachsmuth, P., Kaiser, U., Ruoff, R. S., and Stahlberg, H. Graphene: Substrate preparation and introduction. *J. Struct. Biol.*, 174(1):234–238, 2011. doi:10.1016/j.jsb.2010.10.002.

Patterson, J. P., Sanchez, A. M., Petzetakis, N., Smart, T. P., Epps III, T. H., Portman, I., Wilson, N. R., and O'Reilly, R. K. A simple approach to characterizing block copolymer assemblies: graphene oxide supports for high contrast multi-technique imaging. *Soft Matter*, 8(12):3322–3328, 2012.

Pennington, R. S., Wang, F., and Koch, C. T. Stacked-Bloch-wave electron diffraction simulations using GPU acceleration. *Ultramicroscopy*, 141:32–37, June 2014. doi:10.1016/j.ultramic.2014.03.003.

PresaSoto, A., Gilroy, J. B., Winnik, M. A., and Manners, I. Pointed-Oval-Shaped Micelles from Crystalline-Coil Block Copolymers by Crystallization-Driven Living Self-Assembly. *Angew. Chemie Int. Ed.*, 49(44):8220–8223, 2010. doi:10.1002/anie.201003066.

Reddy, B. and Chatterji, B. An FFT-based technique for translation, rotation, and scale-invariant image registration. *IEEE Trans. Image Process.*, 5(8):1266–1271, 1996.

Rez, D., Rez, P., and Grant, I. DiracâĂŞFock calculations of X-ray scattering factors and contributions to the mean inner potential for electron scattering. *Acta Crystallogr. Sect. A Found. Crystallogr.*, 50(4):481–497, July 1994. doi:10.1107/S0108767393013200.

Rosenauer, A., Schowalter, M., Titantah, J. T., and Lamoen, D. An emission-potential multislice approximation to simulate thermal diffuse scattering in high-resolution transmission electron microscopy. *Ultramicroscopy*, 108(12):1504–13, November 2008. doi:10.1016/j.ultramic.2008.04.002.

Ruskin, R. S., Yu, Z., and Grigorieff, N. Quantitative characterization of electron detectors for transmission electron microscopy. *J. Struct. Biol.*, 184(3):385–93, December 2013. doi:10.1016/j.jsb.2013.10.016.

Sahin, H., Topsakal, M., and Ciraci, S. Structures of fluorinated graphene and their signatures. *Phys. Rev. B*, 83(11):115432, March 2011. doi:10.1103/PhysRevB.83.115432.

Sarvaiya, J., Patnaik, S., and Kothari, K. Image registration using log polar transform and phase correlation to recover higher scale. *J. Pattern Recognit. Res.*, 7:90–105, 2012.

Saxton, W. O. Observation of lens aberrations for very high-resolution electron microscopy. I. Theory. *J. Microsc.*, 179(2):201–213, August 1995. doi:10.1111/j.1365-2818.1995.tb03633.x.

Saxton, W. and Smith, D. J. The determination of atomic positions in high-resolution electron micrographs. *Ultramicroscopy*, 18(1-4):39–47, January 1985. doi:10.1016/0304-3991(85)90120-2.

Scherzer, O. The theoretical resolution limit of the electron microscope. *J. Appl. Phys.*, 20:20–29, 1949.

Schiske, P. Zur frage der bildrekonstruktion durch fokusreihen. In *Proc. Eur. Reg. Conf. Electron. Microsc., 4th*, volume 1, page 145, 1968.

Schiske, P. Image processing using additional statistical information about the object. In Hawkes, P., editor, *Image Process. Comput. Des. Electron Opt.*, volume 94, page 651. Academic Press Inc., 1973.

Schwierz, F. Graphene transistors. *Nat. Nanotechnol.*, 5(7):487–96, July 2010. doi:10.1038/nnano.2010.89.

Shams, R., Sadeghi, P., Kennedy, R., and Hartley, R. Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. *Comput. Methods Programs Biomed.*, 99(2):133–46, August 2010a. doi:10.1016/j.cmpb.2009.11.004.

Shams, R., Sadeghi, P., Kennedy, R., and Hartley, R. Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. *Comput. Methods Programs Biomed.*, 99(2):133–46, August 2010b. doi:10.1016/j.cmpb.2009.11.004.

Shannon, C. E. A Mathematical Theory of Communication. *Bell Syst. Tech. J.*, 27 (4):623–656, October 1948. doi:10.1002/j.1538-7305.1948.tb00917.x.

Spence, J. *High-resolution electron microscopy*. Oxford University Press, Oxford, fourth edition, 2013. ISBN 9780198509158.

Stadelmann, P. A. EMS - a software package for electron diffraction analysis and HREM image simulation in materials science. *Ultramicroscopy*, 21(2):131–145, 1987. doi:http://dx.doi.org/10.1016/0304-3991(87)90080-5.

Stankovich, S., Dikin, D. a., Dommett, G. H. B., Kohlhaas, K. M., Zimney, E. J., Stach, E. a., Piner, R. D., Nguyen, S. T., and Ruoff, R. S. Graphene-based composite materials. *Nature*, 442(7100):282–6, July 2006. doi:10.1038/nature04969.

Takeda, M., Ina, H., and Kobayashi, S. Fourier-transform method of fringe-pattern analysis for computer-based topography and interferometry. *JosA*, 72(1):156–160, 1982.

Talmon, Y. Staining and drying-induced artifacts in electron microscopy of surfactant dispersions. 93(2):366–382, June 1983. doi:10.1016/0021-9797(83)90420-4.

Teague, M. R. Deterministic phase retrieval: a Green's function solution. *JOSA*, 1983. doi:http://dx.doi.org/10.1364/JOSA.73.001434.

Thomas, G. and Lacaze, J.-C. Transmission electron microscopy at 2.5 MeV. *J. Microsc.*, 97(3):301–308, April 1973. doi:10.1111/j.1365-2818.1973.tb03784.x.

Thust, a., Overwijk, M., Coene, W., and Lentzen, M. Numerical correction of lens aberrations in phase-retrieval HRTEM. *Ultramicroscopy*, 64(1-4):249–264, August 1996. doi:10.1016/0304-3991(96)00022-8.

Uhlemann, S. and Haider, M. Residual wave aberrations in the first spherical aberration corrected transmission electron microscope. *Ultramicroscopy*, 72(3-4): 109–119, May 1998. doi:10.1016/S0304-3991(97)00102-2.

Uhlemann, S., Müller, H., Hartel, P., Zach, J., and Haider, M. Thermal Magnetic Field Noise Limits Resolution in Transmission Electron Microscopy. *Phys. Rev. Lett.*, 111(4):046101, July 2013. doi:10.1103/PhysRevLett.111.046101.

Urban, K. W., Jia, C.-L., Houben, L., Lentzen, M., Mi, S.-B., and Tillmann, K. Negative spherical aberration ultrahigh-resolution imaging in corrected transmission electron microscopy. *Philos. Trans. A. Math. Phys. Eng. Sci.*, 367(1903):3735–53, September 2009. doi:10.1098/rsta.2009.0134.

Van Dyck, D. and Coene, W. The real space method for dynamical electron diffraction calculations in high resolution electron microscopy. *Ultramicroscopy*, 15(1-2):29–40, January 1984. doi:10.1016/0304-3991(84)90072-X.

Van Dyck, D. and Chen, F.-R. âĂŸBig BangâĂŹ tomography as a new route to atomic-resolution electron tomography. *Nature*, 486(101):243–246, 2012. doi:10.1038/nature11074.

Viola, P. and Wells III, W. Alignment by maximization of mutual information. *Int. J. Comput. Vis.*, 24(2):1–29, 1997.

Vulović, M., Franken, E., Ravelli, R. B., van Vliet, L. J., and Rieger, B. Precise and unbiased estimation of astigmatism and defocus in transmission electron microscopy. *Ultramicroscopy*, 116:115–134, May 2012. doi:10.1016/j.ultramic.2012.03.004.

Vulović, M., Ravelli, R. B. G., van Vliet, L. J., Koster, A. J., Lazić, I., Lücken, U., Rullgå rd, H., Öktem, O., and Rieger, B. Image formation modeling in cryo-electron microscopy. *J. Struct. Biol.*, 183(1):19–32, July 2013. doi:10.1016/j.jsb.2013.05.008.

Wacker, C. and Schröder, R. R. Multislice algorithms revisited: Solving the Schrödinger equation numerically for imaging with electrons. *Ultramicroscopy*, (0), 2014. doi:http://dx.doi.org/10.1016/j.ultramic.2014.12.008.

Wade, R. H. and Frank, J. Electron-microscope transfer-functions for partially coherent axial illumination and chromatic defocus spread. *Optik (Stuttg).*, 49(1): 81–92, 1977.

Wang, P., Batey, D. J., Rodenburg, J. M., Sawada, H., and Kirkland, A. I. Towards Sub-Angström Ptychographic Diffractive Imaging. *Microsc. Microanal.*, 19 (Supplement S2):706–707, 2013.

Wang, Y., Shi, Z., Huang, Y., Ma, Y., Wang, C., Chen, M., and Chen, Y. Supercapacitor Devices Based on Graphene Materials. *J. Phys. Chem. C*, 113(30): 13103–13107, July 2009. doi:10.1021/jp902214f.

Weickenmeier, a. and Kohl, H. Computation of absorptive form factors for high-energy electron diffraction. *Acta Crystallogr. Sect. A Found. Crystallogr.*, 47(5): 590–597, September 1991. doi:10.1107/S0108767391004804.

Williams, D. B. and Carter, C. B. *Transmission Electron Microscopy: Imaging*. Number v. 1. Plenum Press, 1996. ISBN 9780306453243.

Wilson, N. R., Pandey, P. A., Beanland, R., Rourke, J. P., Lupo, U., Rowlands, G., and Römer, R. A. On the structure and topography of free-standing chemically modified graphene. *New J. Phys.*, 12(12):125010, 2010.

Wolberg, G. and Zokai, S. Robust image registration using log-polar transform. In *Image Process. 2000. Proceedings.*, 2000.

Xu, W., Xu, F., Jones, M., Keszthelyi, B., Sedat, J., Agard, D., and Mueller, K. High-performance iterative electron tomography reconstruction with long-object compensation using graphics processing units (GPUs). *J. Struct. Biol.*, 171(2): 142–153, 2010. doi:10.1016/j.jsb.2010.03.018.

Yang, Y. Q., Zheng, L. S., Guo, X. D., Qian, Y., and Zhang, L. J. pH-Sensitive Micelles Self-Assembled from Amphiphilic Copolymer Brush for Delivery of Poorly Water-Soluble Drugs. *Biomacromolecules*, 12(1):116–122, December 2010. doi:10.1021/bm101058w.

Zemlin, F., Weiss, K., Schiske, P., Kunath, W., and Herrmann, K. Coma-free alignment of high resolution electron microscopes with the aid of optical diffractograms. *Ultramicroscopy*, 3:49–60, 1978.

Zernike, F. Phase contrast, a new method for the microscopic observation of transparent objects. *Physica*, (7):686–698, 1942. doi:http://dx.doi.org/10.1016/S0031-8914(42)80035-X.

Zhang, L. F. and Eisenberg, A. Multiple Morphologies of "Crew-Cut" Aggregates of Polystyrene-b-poly(acrylic acid) Block Copolymers. *Science (80-.).*, 268(5218): 1728–1731, 1995.

Acronyms

ADU analog-to-digital unit. 48

AFM atomic force microscopy. 81, 132

ALU arithmetic logic unit. 16

CBED convergent beam electron diffraction. 40, 65

CCD charge-coupled device. vii, xiii, xviii, 4, 5, 12, 14, 26, 40, 47, 48, 57, 61, 76, 94, 110, 139

CMS conventional multislice. vii, viii, 39, 40, 42, 46, 56–60, 62, 64, 65, 151

CPU central processing unit. ix–xi, xviii, 14–16, 18–20, 39, 49, 51, 54, 62, 64, 65, 67–69, 94, 96, 103–106, 108, 159

DAC digital to analog converter. 33, 36, 74, 90, 91

DFT density functional theory. vii, xii, 15, 42, 43, 128

DFTR discrete fourier transform. 85

DM Digital Micrograph^{TM}. xi, 72, 93–95, 107–109

DQE detective quantum efficiency. vii, 46–48

EELS electron energy loss spectroscopy. 75

EWR exit wave reconstruction. xi–xiv, xviii, 13, 20–22, 29, 30, 32–34, 37, 71, 72, 76, 77, 84, 89, 91–95, 108–110, 113, 119–121, 123, 126, 128–130, 132, 134–136, 138, 140, 144–146, 148, 149

FDMS finite difference multislice. viii, 56–60, 62, 65

FFT fast Fourier transform. 22, 39, 55, 60, 77, 78, 84, 85, 95, 107, 116, 138

FRWR full resolution wavefunction reconstruction. 32

FTSR focal and tilt series reconstruction. 29, 32, 46, 72, 89, 92–94, 101, 102, 109, 141

GO graphene oxide. xiv, 132–134, 137, 142–145

GPU graphics processing unit. v, vii–xi, xviii, 13–21, 27, 39, 40, 49, 51–56, 61–69, 77, 93–96, 98, 100, 102–108, 110, 148–152, 156, 157, 159, 160

GUI graphical user interface. ix, 70, 74, 107, 108

HOLZ higher-order laue zone. 57

HRTEM high resolution transmission electron microscopy. xii, 5, 25, 120, 125, 132, 133, 137, 144

IMS improved multislice. vii, viii, 42, 56–60, 62, 65, 69, 151, 155

IWFR iterative wave function reconstruction. 32, 89, 92

LUT lookup table. 54, 55

MAL maximum-likelihood. 89, 92

MI mutual information. xii, xviii, 77, 80–83, 86, 88, 93, 96, 98, 109, 110, 112, 121–123

MTF modulation transfer function. xiii, 46, 47, 139

NCC normalized cross correlation. 91, 100

NNPS normalized noise power spectrum. 47

NPS noise power spectrum. 46, 47

NTF noise transfer function. 47, 48

PCF phase correlation function. ix, 78, 79, 84, 95

PCI phase contrast index. 31, 102, 103

PCPCF phase compensated phase correlation function. ix, x, 77–79, 83, 87, 89, 93, 95, 96, 103, 104, 111, 122, 160

PCTF phase contrast transfer function. ix, 8, 10, 12, 33, 71–74, 133, 135, 136

PSD power spectral density. 135, 136

RAM random access memory. 51

RSMS real space multislice. 39

SAED selected area electron diffraction. 4

SDK software development kit. 107, 108

SEM scanning electron microscope. 81, 131

SNR signal to noise ratio. 46, 99, 101

STEM scanning transmission electron microscopy. 2, 40, 65

SVD singular value decomposition. 90

TEM transmission electron microscopy. vi, xviii, 1–5, 7, 9, 13, 21, 22, 26, 32–34, 37, 38, 42, 46, 69, 71, 72, 75, 81, 86, 91, 130–132, 145, 149

WPOA weak phase object approximation. 6

XCF cross correlation function. ix, 77–79, 83, 84